# Oracle® Retail Merchandising System

Operations Guide Addendum
Release 11.0.12

August 2007

ORACLE

Oracle® Merchandising System Operations Guide Addendum, Release 11.0.12



Primary Author: Nathan Young

## Value-Added Reseller (VAR) Language

(i) the software component known as **ACUMATE** developed and licensed by Lucent Technologies Inc. of Murray Hill, New Jersey, to Oracle and imbedded in the Oracle Retail Predictive Application Server – Enterprise Engine, Oracle Retail Category Management, Oracle Retail Item Planning, Oracle Retail Merchandise Financial Planning, Oracle Retail Advanced Inventory Planning and Oracle Retail Demand Forecasting applications.

(ii) the **MicroStrategy** Components developed and licensed by MicroStrategy Services Corporation (MicroStrategy) of McLean, Virginia to Oracle and imbedded in the MicroStrategy for Oracle Retail Data Warehouse and MicroStrategy for Oracle Retail Planning & Optimization applications.

(iii) the **SeeBeyond** component developed and licensed by Sun MicroSystems, Inc. (Sun) of Santa Clara, California, to Oracle and imbedded in the Oracle Retail Integration Bus application.

(iv) the **Wavelink** component developed and licensed by Wavelink Corporation (Wavelink) of Kirkland, Washington, to Oracle and imbedded in Oracle Retail Store Inventory Management.

(v) the software component known as **Crystal Enterprise Professional and/or Crystal Reports Professional** licensed by Business Objects Software Limited ("Business Objects") and imbedded in Oracle Retail Store Inventory Management.

(vi) the software component known as **Access Via™** licensed by Access Via of Seattle, Washington, and imbedded in Oracle Retail Signs and Oracle Retail Labels and Tags.

(vii) the software component known as **Adobe Flex™** licensed by Adobe Systems Incorporated of San Jose, California, and imbedded in Oracle Retail Promotion Planning & Optimization application.

(viii) the software component known as **Style Report™** developed and licensed by InetSoft Technology Corp. of Piscataway, New Jersey, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

(ix) the software component known as **i-net Crystal-Clear™** developed and licensed by I-NET Software Inc. of Berlin, Germany, to Oracle and imbedded in the Oracle Retail Central Office and Oracle Retail Back Office applications.

(x) the software component known as **WebLogic™** developed and licensed by BEA Systems, Inc. of San Jose, California, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

(xi) the software component known as **DataBeacon™** developed and licensed by Cognos Incorporated of Ottawa, Ontario, Canada, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

# Contents

# Preface

Oracle Retail Operations Guides are designed so that you can view and understand the application's 'behind-the-scenes' processing, including such information as the following:

- Key system administration configuration settings
- Technical architecture

## Audience

Anyone with an interest in developing a deeper understanding of the underlying processes and architecture supporting RMS functionality will find valuable information in this guide. There are three audiences in general for whom this guide is written:

- Business analysts looking for information about processes and interfaces to validate the support for business scenarios within RMS and other systems across the enterprise.
- System analysts and system operations personnel:

  Who are looking for information about RMS processes internally or in relation to the systems across the enterprise.

  Who operate RMS regularly.
- Integrators and implementation staff with overall responsibility for implementing RMS.

## Related Documents

For more information, see the following documents in the Oracle Retail Merchandising System Release 11.0.12 documentation set:

- Oracle Retail Merchandising System Installation Guide
- Oracle Retail Merchandising System Release Notes
- Oracle Retail Merchandising System Data Model
- Oracle Retail Merchandising System Batch Schedule

## Customer Support

https://metalink.oracle.com

When contacting Customer Support, please provide the following:

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

## Review Patch Documentation

For a base release (".0" release, such as 12.0), Oracle Retail strongly recommends that you read all patch documentation before you begin installation procedures. Patch documentation can contain critical information related to the base release, based on new information and code changes that have been made since the base release.

## Oracle Retail Documentation on the Oracle Technology Network

In addition to being packaged with each product release (on the base or patch level), all Oracle Retail documentation is available on the following Web site:

http://www.oracle.com/technology/documentation/oracle_retail.html

Documentation should be available on this Web site within a month after a product release. Note that documentation is always available with the packaged code on the release date.

## Conventions

**Navigate:** This is a navigate statement. It tells you how to get to the start of the procedure and ends with a screen shot of the starting point and the statement "the Window Name window opens."

> **Note:** This is a note. It is used to call out information that is important, but not necessarily part of the procedure.

```
This is a code sample
    It is used to display examples of code
```

A hyperlink appears like this.

# 1
# Introduction

## Overview

The information in this document reflects modifications and updates to the *Oracle Retail Merchandising System 11.0 Operations Guide* and any subsequent RMS 11.0.x Operations Guide Addendums. (The RMS 11.0 Operations Guide is the most recent release of the full Operations Guide for the 11.0 release of RMS.) Using this document in conjunction with the *Oracle Retail Merchandising System 11.0 Operations Guide* provides retailers with a complete overview of the application.

For more specific information regarding enhancements and modifications made to the previous Oracle Retail Merchandising System release, see the Oracle Retail Merchandising System 11.0.12 Release Notes.

# 2

# Batch Designs

Retailers should refer to these sections in lieu of the corresponding batch designs in the RMS 11.0 Operations Guide or any subsequent RMS 11.0.x Operation Guide Addendums.

Batch designs describe how, on a technical level, an individual batch module works and the database tables that it affects. In addition, batch designs contain file layout information that is associated with the batch process.

## Sales Audit ACH Download [saexpach]

### Functional Area

Sales Audit Export – Automated Clearing House (ACH)

### Module Affected

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 |
| 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.

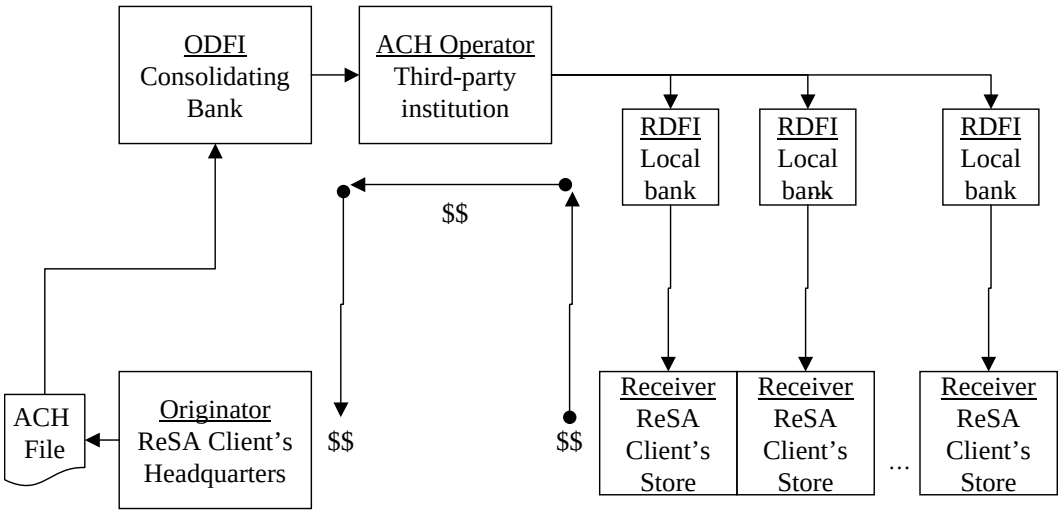



**Figure 1: Overview of an ACH Network**

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. Oracle Retail 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. This module should not be run simultaneously with other modules: saexprms, saexprdw, saexpim, saexpuar, and saexpgl.

Threading Scheme: N/A

## Restart Recovery

Logical Unit of Work (recommended Commit checkpoints)

Driving Cursor

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

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

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

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

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

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

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

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

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

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

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

## Program Flow

Structure Chart

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

Functional Design – SA export.doc

## Shared Modules

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

**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

All database interactions required and error handling considerations

**Init ()**

- Initialize restart/recovery by calling restart_init().
- Get the vdate from the period table and the system time.

- Get the system level information: the sender id, the company id, the consolidating bank name, the consolidating routing number and the consolidating account number. These are on the sa_ach_info table.

**Process ()**

1. Get the next store/day to be processed (exported) by fetching from the first driving cursor.
2. Attempt to lock the store/day with a call to **get_lock()**. If this fails, write the store to a linked list (which contains all unprocessed store/days).
3. Skip to step 7 if the store of the store/day to be processed is for a store which is on the linked list.
4. Call the function **postStoreACH()** for the current store/day.
5. Set sa_export_log.status to SAES_P by calling **setProcessed()** for the current store/day, so that it will not be processed again in case of a restart.
6. Call **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()**.
8. If the amount to be transferred is negative, the record should be skipped.
9. If the account is a checking account, the transaction code to use is '27'.
10. If the account is a savings account, the transaction code to use is '37'.
11. If the amount to be transferred is positive, call the function **PostBankACHTable()** to record the amount of the ACH entry, else do nothing.
12. Keep running totals for the current batch's total amount and the total ACH amounts.
13. Commit after pl_commit_max_ctr LUW have been processed. Redefine the SAVEPOINT after the commit because savepoints are lost after a commit.
14. Loop to step 3 until the cursor returns no data.
15. Write the ACH Batch Control record and the ACH File Control record
16. 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.
17. Mark all store/days *that were not locked* (i.e. those with a sa_export_log.status of SAES_P) as completed (SAES_E) in the sa_export_log.

**postBankACHTable ()**

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

1. If an entry does not exist for the current bank and date in the sa_bank_ach table:
   - Make an entry in the sa_bank_ach table for the current bank and account placing the sums of the store ACH amounts and adjustments in the ACH amount field (sa_bank_ach.ach_amt).
2. If an entry exists for the current bank and date in the sa_bank_ach table:
   - Add the manual adjustment to the bank ACH deposit amount.
   - Update the sa_bank_ach table with the bank ACH deposit amount (sa_bank_ach.ach_amt).

**File Output Functions**

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

**Linked List Functions**

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

**MarkAllStoreDaysCompleted ()**

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

**SetCurrencyDecimals ()**

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

**TruncateDec ()**

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

# I/O Specification

## 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_OPTIONS. ACH_SENDER_ID | 10 | None |
| File Creation Date | Date when the file was created. | YYMMDD | 6 | None |
| File Creation Time | Time when the file was created. | HH24MM | 4 | None |

| Field Name | Field Description | Value | Length | Jstf/ Pad* |
|---|---|---|---|---|
| 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.<br>CONSOL_BANK_NAME | 23 | L/B |
| Immediate Origin Name | The name of the company. | COMPHEAD.<br>CO_NAME | 23 | L/B |
| Reference Code | Any reference code. This is an optional field. ReSA will not populate this field as the create datetime should be enough to reference the data that was exported by comparing with SA_EXPORTED.<br>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.<br>CO_NAME | 16 | L/B |
| Company Discretionary Data | Any kind of data specific to the company. ReSA will not use this field | blanks | 20 | None |
| Company Identification | An alphanumeric code identifying the company. The first character may be the ANSI one-digit Identification Code Designators (ICD). For example,<br>"1" IRS Employer ID Number<br>"9" User Assigned Number.<br>ReSA assumes that the company_id field on the sa_system_options table will contain the correct id. | SA_SYSTEM_OPTIONS.<br>COMPANY_ID | 10 | L/B |
| Standard Entry Class Code | This provides a way to distinguish between the various kinds of entries. Since ReSA will be sending CCD entries, this field should hold the value 'CCD'. | 'CCD' | 3 | None |
| Company Entry Description | A short description from the Originator about the purpose of the entry. | 'CONSOL.' | 10 | L/B |
| Company Descriptive Date | Optional field providing a date to the Receiver for descriptive purposes. ReSA will populate it with the next day's date in the YYMMDD format. | YYMMDD format of<br>PERIOD.VDATE + 1 | 6 | None |
| Effective Entry Date | The date by which the Originator intends the batch of entries to be settled. Since the Originator will want this to be done as soon as possible, ReSA will use the earliest possible date, which is one banking day after the processing date (the current date). | YYMMDD format of<br>PERIOD.VDATE + 1 | 6 | None |
| Settlement Date | This is inserted by receiving ACH Operator. ReSA will leave this blank. | blanks | 3 | None |
| Originator Status Code | This field stores a code to describe the type of Originator. This should be a 1 to describe the Originator as a depository financial institution. | '1' | 1 | None |

| Field Name | Field Description | Value | Length | Jstf/ Pad* |
|---|---|---|---|---|
| ODFI Identification | 8-digit routing number of the ODFI. | First 8 digits of SA_BANK_STORE.<br>CONSOLIDATING_ROUTING_NO | 8 | None |
| Batch Number | The batch number. | | 7 | R/0 |

**Batch Header Record for CBR transactions**

| Field Name | Field Description | Value | Length | Jstf/ Pad* |
|---|---|---|---|---|
| Record Type Code | The type of record. | '5' | 1 | None |
| Service Class Code | This field identifies the general classification of dollar entries to be exchanged. Funds will always flow from the local banks to the consolidating bank. Hence the code '225' for "ACH Debits only" should be used. | '225' | 3 | None |
| Company Name | The name of the company. | First 16 characters of COMPHEAD.<br>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,<br>"1" IRS Employer ID Number<br>"9" User Assigned Number.<br>ReSA assumes that the company_id field on the sa_system_options table will contain the correct id. | SA_SYSTEM_OPTIONS.<br>COMPANY_ID | 10 | L/B |
| Standard Entry Class Code | This provides a way to distinguish between the various kinds of entries. Since ReSA will be sending CBR entries, this field should hold the value 'CBR'. | 'CBR' | 3 | None |
| Company Entry Description | A short description from the Originator about the purpose of the entry. | 'CONSOL.' | 10 | L/B |
| ISO Originating Currency Code | Currency code in which the funds are originating from. This must be the ISO code of the currency. | PARTNER.<br>CURRENCY_CODE | 3 | None |
| ISO Destination Currency Code | Currency code in which the funds are to be received. This must be "USD". | 'USD' | 3 | None |
| Effective Entry Date | The date by which the Originator intends the batch of entries to be settled. Since the Originator will want this to be done as soon as possible, ReSA will use the earliest possible date, which is one banking day after the processing date (the current date). | YYMMDD format of<br>PERIOD.VDATE + 1 | 6 | None |
| Settlement Date | This is inserted by receiving ACH Operator. ReSA will leave this blank. | blanks | 3 | None |
| Originator Status Code | This field stores a code to describe the type of Originator. This should be a 1 to describe the Originator as a depository financial institution. | '1' | 1 | None |
| ODFI Identification | 8-digit routing number of the ODFI. | First 8 digits of SA_BANK_STORE.<br>CONSOLIDATING_ROUTING_NO | 8 | None |
| Batch Number | The batch number. It is not expected that the file will have more than two batches. | '1' or '2' | 7 | R/0 |

**CCD Entry Detail Record**

| Field Name | Field Description | Value | Length | Jstf/ Pad* |
|---|---|---|---|---|
| Record Type Code | The type of record. | '6' | 1 | None |
| Transaction Code | Code used to identify the type of debit and credit. This is dependent on the type of account and on the direction of funds transfer.<br>'27' – if the account is a checking account,<br>'37' – if the account is a savings account. | '27' or '37' | 2 | None |
| RDFI Identification | 8-digit routing number of the RDFI. | First 8 digits of SA_BANK_STORE.<br>ROUTING_NO | 8 | None |
| Check Digit | This is the 9th digit from the routing number. | 9th digit of SA_BANK_STORE.<br>ROUTING_NO | 1 | None |
| DFI Account Number | The account number at the local bank. | SA_BANK_STORE.<br>BANK_ACCT_NO | 17 | L/B |
| Amount | The amount involved in the transaction. This field is numeric only and the last two digits are automatically assumed to be decimals. ReSA amounts are stored as 20 digit numbers, with 4 for decimals. ReSA will truncate the last two digits of the amount and should the resulting amount be greater than 10 digits, this program will abort with an error. It is not expected that a client will send an ACH amount greater than US$100 million. The values for this are taken from the sa_store_ach table. The values from the columns today_adj_deposit_est, next_day_man_adj_deposit, and next_day_deposit_est are added up by business_date and then multiplied by 10000 and later divided by 100 to obtain a dollar amount. | | 10 | R/0 |
| Identification Number | Optional field containing a number used by Originator to insert its own number for tracing purposes. ReSA will not populate this field. | blanks | 15 | None |
| Receiving Company Name | Name of the local store. | STORE.<br>STORE_NAME | 22 | L/B |

| Field Name | Field Description | Value | Length | Jstf/ Pad* |
|---|---|---|---|---|
| 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.<br>'27' – if the account is a checking account,<br>'37' – if the account is a savings account. | '27' or '37' | 2 | None |
| RDFI Identification | 8-digit routing number of the RDFI. | First 8 digits of SA_BANK_STORE.<br>ROUTING_NO | 8 | None |
| Check Digit | This is the 9th digit from the routing number. | 9th digit of SA_BANK_STORE.<br>ROUTING_NO | 1 | None |
| DFI Account Number | The account number at the local bank. | SA_BANK_STORE.<br>BANK_ACCT_NO | 17 | L/B |

| Field Name | Field Description | Value | Length | Jstf/ Pad* |
|---|---|---|---|---|
| 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 |
| 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 |

| Field Name | Field Description | Value | Length | Jstf/ Pad* |
|---|---|---|---|---|
| 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 |
| 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,<br>"1" IRS Employer ID Number<br>"9" User Assigned Number.<br>ReSA assumes that the company_id field on the sa_system_options table will contain the correct id. | SA_SYSTEM_OPTIONS.<br>COMPANY_ID | 10 | L/B |

| Field Name | Field Description | Value | Length | Jstf/ Pad* |
|---|---|---|---|---|
| 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.<br>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 |

# Sales Audit Export to GL [saexpgl]

## Design Overview

The purpose of this batch module is to post all properly configured user defined ReSA totals to the User defined General ledger application (Oracle or PeopleSoft). Totals without errors will be posted to the appropriate accounting ledger, as defined in the Sales Audit Oracle cross-reference user module. Depending on the unit of work system option, the data will be sent at either the store day or individual total level. Newly revised totals that have already been posted to the ledger will have their previous revision reversed, and the new total posted to the appropriate accounts. Transactions that are from previous periods will be posted to the current period.

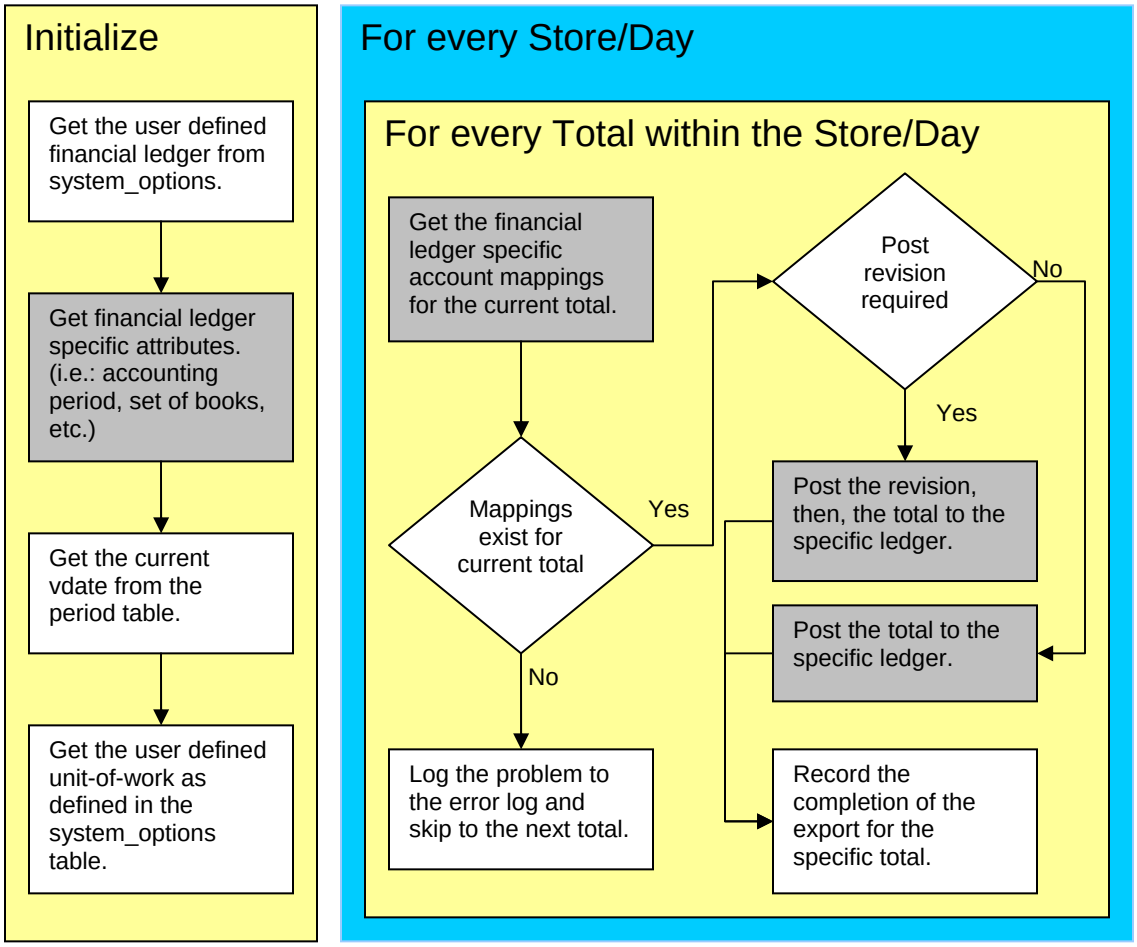
This version of the program is meant for the interface between RMS 11.0 and Oracle Financials.

## Tables Affected

| TABLE | SELECT | INSERT | UPDATE | DELETE |
|---|---|---|---|---|
| period | Yes | No | No | No |
| sa_system_options | Yes | No | No | No |
| sa_store_day | Yes | No | No | No |
| sa_export_log | Yes | No | Yes | No |
| sa_error | Yes | No | No | No |
| sa_exported | Yes | Yes | No | No |
| sa_balance_group | Yes | No | No | No |
| sa_error_rev | Yes | No | No | No |
| sa_exported_rev | Yes | No | No | No |
| sa_store_day_lock | Yes | Yes | No | Yes |
| fif_gl_setup | Yes | No | No | No |
| store | Yes | No | No | No |
| sa_fif_gl_cross_ref | Yes | No | No | No |
| stg_fif_gl_data | No | Yes | No | No |
| if_errors | No | Yes | No | No |

## Program Flow

Below is a simple flow of the general ledger export and its generic and financial application specific modules:




**Legend:**

Process specific to user defined financial application.

Process generic to all financial applications.

## Global Variable Descriptions

| Global Variable | Description |
|---|---|
| pi_commit_max_ctr | Commit max counter used for array fetch |
| ps_num_threads | Commit max counter used for array fetch |
| ps_thread_val | Commit max counter used for array fetch – Thread value |
| pi_proc_cnt | Commit max counter used for array fetch |
| ps_sysdate | Current sysdate value from the database. |
| ps_store_day_seq_no | Restart/recovery variables used for bookmarking |
| ps_vdate | Date value from the period table |
| ps_unit_of_work | Unit of Work from sa_system_options. |
| ps_update_id | Update ID from fif_gl_setup |
| ps_set of books_id | Set of Books ID from fif_gl_setup |
| ps_period | Period Name from fif_gl_setup |
| pi_num_locks_not_released | Counter for the number of store/day locks that could not be released. |
| pi_rec_ctr | Counter for the number of records processed and inserted to stg_fif_gl_data table.. |
| pi_non_fatal | Counter for the number of non-fatal errors encountered. |

## Function Level Description

**main()**

1. Check command line for required arguments.
2. Call **LOGON** to connect to the database.
3. Call **Init** to initialize the program.
4. Call **process** to export the available RMS data.
5. Report unlocking errors.
6. Call **final** to cleanup.

**init()**

1. Call retek_init.
2. Get the current vdate from the period table, using fetchVdate.
3. Get the user financial application type from system_options.financial_app.
   'O' = Oracle GL
   'P' = PeopleSoft GL
4. Get the Financial application specific attributes (i.e. accounting period information, set of books identntifier, etc.)

5. If Oracle GL, retrieve the following details as defined in the RMS database:
   Fif_gl_setup.set_of_books_id
   Fif_gl_setup.last_update_id
6. Get and save the value of sa_system_options.unit_of_work, by calling the function **fetchSaSystemOptions.**

#### process()

1. Retrieve a store/day by calling **fetchStoreDayToBeExported**.
2. Attempt to lock the store/day with a call to **get_lock**. If this fails, go on to the next store/day.
3. Find out the number of errors pending for the store/day by calling **fetchStoreDayErrorCount.**
4. If the unit of work is store and the number of errors in the store/day is greater than zero, then release the lock by calling **release_lock** and skip the store/day, otherwise continue.
5. Retrieve a total to export by calling **getTotal**.
6. If Oracle GL, check to ensure that the selected total has a user defined cross-reference in the sa_fif_ora_cross_ref table by calling the function **getOracleMapping**. If a mapping (Oracle CCID) does not exist for the selected total log the problem in the Retek error log and go onto the next total.
7. If the tran_sign is 'N' (code_type is SAFD), the currenct retrieved value will be post to Oracle with negative sign.
8. Post the current total to the GL by calling the financial application specific function:
9. If Oracle, call **postOracleGL**
10. If there are more totals for the selected store/day, loop through the store day totals (getTotal).
11. Call the library function **markStoreDayExported**.
12. Call **release_lock** and go on to the next store/day.

#### ProcessStoreDay()

1. Get all the totals for the store/day by calling **getTotal().**
2. For each Total_id, call **getOracleMapping()** for Oracle account.
3. If Status returned from getTotal() is 'N'. The opposite amounts will be posted to the Stg_fif_gl_data table (that is, send a negative number).
4. Call **UpdateGLArray()** to populate gl_data_array for inserting stg_fif_gl_data table.
5. Call the library function **markTotalExported** and include the current period number. This function has to be called once for each total that is exported.

#### CanProcess()

1. Calling **fetchStoreDayErrorCount** to find out the number of errors pending for the store/day.
2. If the unit of work is store and the number of errors in the store/day is greater than zero, skip the store/day and write to the if_errors for the store/day.

#### final()

1. Clean up – free any memory used.
2. Call retek_close.

#### AddToList()

Setup linked list to hold locked store/day for later process.

#### DeleteList()

This function deletes linked list, and free the memory.

#### GetNext()

This function moves the pointer to the next unprocessed store/day.

#### RemoveFromList()

This function removes processed store/day from linked list.

#### SizeGlDataArray()

This function allocates memory for gl_data_array.

#### ProcessLockedSD()

This function locks the store/day to be processed.

#### GetOracleMapping()

This function will load local variables with the user-defined accounts and CCID's for the selected total/location combination from the SA_FIF_GL_CROSS_REF table. If no results are returned, the total should be skipped with the appropriate message in the Retek error log.

#### InsertToOracleGL()

This function inserts the record processed into STG_FIF_GL_DATA table.

#### UpdateGLArray()

This function writes store/day total to the gl_data array for inserting to stg_fif_gl_data. Post the current total using the mapped local variables retrieved from the **getOracleMapping** function. First insert a record for the debit side of the transaction, then insert a record for the credit half of the transaction. (See STG_FIF_GL_DATA details below). The following is a detailed explanation of the required columns in the Oracle STG_FIF_GL_DATA table.

**STG_FIF_GL_DATA column explanation**

| Column | Description |
|---|---|
| status | This column represents the type of posting being applied. All inserts from this module, status should be set to 'NEW'. |
| set_of_books_id | This column represents the identifier for the book of accounts that this module will be posting to. This field should always be set to the value found in FIF_GL_SETUP.SET_BOOKS_ID |
| accounting_date | The date of the transaction/total – SA_STORE_DAY.BUSINESS_DATE. |
| currency_code | The default system currency code |
| date_created | period.vdate |

| Column | Description |
|---|---|
| created_by | This field represents the identifier of the application/user whom created this journal entry. This value should be populated with the FIF_GL_SETUP.LAST_UPDATED_ID. |
| actual_flag | The hard-coded value 'A' will represent actual amounts. |
| user_je_category_name | Journal entry source name for the posted transaction. This entry must exist in the Oracle USER_JE_CATEGORY_NAME column in the Journal Categories table prior to posting data to the GL. This value should be hard-coded to 'ReSA'. |
| user_je_source_name | Journal entry source name for the posted transaction. This entry must exist in the Oracle USER_JE_SOURCE_NAME column in the Journal Sources table prior to posting data to the GL. This value should be hard-coded to 'ReSA'. |
| currency_conversion_date | The date in which the total was converted to the default currency code. This value should be populated with the store day bussiness date. |
| currency_conversion_type | This value should be hard-coded to 'Spot'. |
| segment1 – 10 | These columns should be populated with either the debit segment values or the credit values (depending on which half of the total you are posting). |
| entered_dr_amount | If you are entering the debit half of the total, place the total amount in this column. If you are representing the credit half of the total, place a 0 in this column. |
| entered_cr_amount | If you are entering the credit half of the total, place the total amount in this column. If you are representing the debit half of the total, place a 0 in this column. |
| period_name | This value should be populated with the FIF_GL_SETUP.PERIOD_NAME. |
| code_combination_id | If this is the debit half of the total adjustment, place the SA_FIF_GL_CROSS_REF.DR_CCID. If this is the credit half of the total adjustment, place the SA_FIF_GL_CROSS_REF.CR_CCID. |

#### WriteErrorTable()

This function writes to if_errors when error is encountered while inserting to Oracle tables.

#### Stored Procedures / Shared Modules (Maintainability)

| Shared Module | Module Description |
|---|---|
| libresa.a | ReSA Library |
| get_lock | used to establish a read lock on a store/day |
| release_lock | used to release a store/day lock |
| fetchStoreDayToBeExported | This fetches all store days that are ready for export for a given usage type. |

| Shared Module | Module Description |
|---|---|
| getTotal | This fetches all totals that can be exported for the given usage type and for the given store day. |
| fetchStoreDayErrorCount | This functions returns the number of errors pending for a given store day. |
| markTotalExported | records the passed total as exported |
| markStoreDayExported | records the passed store day as exported |
| fetchSaSystemOptions | This function retrieves all entries in the sa_system_options table. |
| fetchVdate | This function retrieves the vdate from the period table. |

Refer to the following documents for more details on the export library:

| Shared Module | Module Description |
|---|---|
| Library Design | saexplib.doc. |
| libretek.a | Retek Library |
| retek_init | initialize restart/recovery |
| retek_close | finalize restart/recovery |
| LANGUAGE_SQL.GET_CODE_DESC | This function will retrieve the description of the passed in code and code type. |

## Input/Output Specifications

There are no input or output files for this export. All data is retrieved from ReSA database tables (as listed above) and posted to the Oracle GL staging table STG_FIF_GL_DATA or the PeopleSoft staging table PS_CPI_GL_DATA.

## Integrity Constraints

Processing Cycle: Anytime – Sales Audit 11.0 is a 24/7 system.

Scheduling Diagram: This program will be run after the ReSA totaling process: satotals.pc and sarules.pc. This module should not be run simultaneously with other modules: saexprms, saexprdw, saexpim, saexpuar, and saexpach.

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 pi_commit_max_ctr. Only one commit will be done: at the end, after a store/day has been completely processed, a call to release_lock() performs a commit.

There are 2 driving cursors in this module. The first picks a store/day to work on. The second fetches the totals to be posted for the store/day.

#### Driving cursor 1:

This driving cursor is embedded in the library function fetchStoreDayToBeExported(). Given a system code, of 'SYSE', this function fetches all store/days with a store_status of

'C'lose, a data_status of 'F'ully loaded and an audit_status of 'A'udited, 'S'tore errors pending or 'H'Q errors pending that are ready to export to the given system.

**Driving cursor 2:**

This driving cursor is embedded in the library function getTotal(). Given a store_day_seq_no and a usage type of 'SAYT', this function retrieves all totals.

# Sales Audit Export to ReIM [saexpim]

## Design Overview

The purpose of this program is to support invoices from Direct Store Delivery and Escheatment sales audit transactions. Direct Store Delivery invoices refer to products or services that are delivered to the store and paid out at the store. This program will take DSD invoices that have been staged to the SA_TRAN_HEAD table by the saimptlog.pc program and move them into the INVC_HEAD table. All DSD transactions will be assumed paid. They can be assumed received if there is a proof of delivery number listed on them. Transactions with a vendor invoice ID or a proof of delivery number should be matched to any existing invoice in INVC_HEAD, and that invoice updated with the new information being interfaced. Invoices that do not match an existing invoice in INVC_HEAD will need to be inserted. Each transaction will be exported to INVC_HEAD table only once.

The Sales Audit Transaction type used to identify invoices for Direct Store Delivery transactions will be "Paid Out". Transaction types are stored on the codes tables with a code_type = 'TRAT'. The Paid Out transaction has a code of 'PAIDOU'. The Sales Audit sub-transaction types will be used to identify whether the invoice is an "Expense Vendor Payout" or a "Merchandise Vendor Payout". These types are stored on the codes table with a code_type = 'TRAS'. The codes will be 'EV' for Expense Vendor Payout and 'MV' for Merchandise Vendor Payout. Any Paid Out transaction with a sub transaction type of Expense Vendor will create a non-merchandise invoice and cause a record to be written to the INVC_NON_MERCH table. ReSA will store non-merchandise codes in the reason_code field on sa_tran_head. Valid values for these reason codes should correspond to the codes stored on the non_merch_code_head table.

In addition to DSD invoices, this program will also interface Escheatment totals to Invoice Matching. Escheatment is the process where an unredeemed gift certificate/voucher or credit voucher will, after a set period of time, be paid out as income to the issuing Retailer or in some states, the State receives this escheatment income. ReSA will be the governing system that determines who receives this income, but Invoice Matching will send the totals, with the related Partner, to Accounts Payable. Escheatment information will be stored on the ReSA SA_TOTALS table and will be used to create non-merchandise invoices in Invoice Matching. These invoices will be assumed not paid.

To accommodate Escheatment, a new calculation should be added to Sales Audit to create escheatment totals. ReSA automatically totals sales transactions based on calculation definitions that the customer's users have previously created using the online wizard. Whenever users create new calculation definitions or edit existing ones, they become part of the automated totaling process the next time that satotals.pc runs.

| Table | Operations Performed | | | |
|---|---|---|---|---|
| | Select | Insert | Update | Delete |
| Period | Yes | No | No | No |
| Sa_system_options | Yes | No | No | No |
| Sa_export_options | Yes | No | No | No |
| Sa_store_day | Yes | No | No | No |
| Sa_store_day_read_lock | Yes | Yes | No | Yes |
| Sa_export_log | Yes | No | Yes | No |
| Sa_tran_head | Yes | No | No | No |
| Sa_tran_tender | Yes | No | No | No |
| Sa_exported | No | Yes | No | No |
| Sa_exported_rev | No | Yes | No | No |
| Sa_total | Yes | No | No | No |
| Sa_error | Yes | No | No | No |
| Invc_head | Yes | Yes | Yes | No |
| Invc_non_merch | Yes | Yes | Yes | No |
| Terms | Yes | No | No | No |
| Currency_rates | Yes | No | No | No |
| Addr | Yes | No | No | No |

## Stored Procedures / Shared Modules (Maintainability)

### libretek library functions:

**retek_init/retek_close/retek_refresh_thread -** to initialize and close Retek's restart/recovery module

### libresa library functions:

- **fetchSysdate –** to get the current date and time
- **fetchSaSystemOptions –** to get ReSA system options
- **fetchStoreDayErrorCount –** to determine whether there are errors for the current store_day
- **markStoreDayExported –** to mark a store day as updated
- **updateStoreDayExported –** to force commit of status 'E' on sa_export_log for store days that have been exported
- **getTotal –** to fetch totals for escheatment invoices
- **get_lock** – to establish a read lock on a store_day
- **release_lock** – to release a store_day lock

## Packages:

- **INVC_SQL.NEXT_INVC_ID**—generates a new invoice id
- **DIRECT_STORE_INVC_SQL.CHECK_INVC_DUPS** – Checks if invoice exists.

## Input Specifications

### 'Table-To-Table'

Foundation tables used by this program for reference include:

Sa_store_data

Sa_system_options

Sa_export_options

Sa_error_codes

Sa_error_impact

Transaction data tables used to drive the processing includes:

Sa_tran_head

Sa_total

Sa_tran_tender

Sa_exported

Sa_error

**Sa_total_usage** – Add new Invoice Matching code ('IM') to the code_detail table where code_type = 'SAUT' (Sales Audit Usage Type). This code will be used as the invoice matching usage_type. Using the ReSA on-line wizard, add a new 'Total' calculation for escheatment transactions.

| Total_id | Usage_type |
|---|---|
| ESCHEAT | IM |

**Sa_store_data** – add invoice matching code to code_detail table where code_type = 'SYSE'

### Code_detail

| Code_type | Code | Code_desc | Code_seq |
|---|---|---|---|
| SYSE | IM | IM Export | 7 |

Sa_store_data – For each store that will require the Invoice Matching Export, add a record to the sa_store_data table with a system_code of 'IM' (Invoice Matching) and imp_exp of 'E' (Export).

| Store | System_code | Imp_exp |
|---|---|---|
| e.g. 1009 | IM | E |

**Sa_export options** – add IM code to the export options table to identify this new export to Invoice Matching. Since this program will not follow full disclosure logic, the multiple_export_ind should be 'N' (No).

| System_code | Multiple_export_ind | Exp_detail_ind |
|---|---|---|
| SYSE | N | IM Export |

## Output Specifications

### 'Table-To-Table'

Updates/Inserts made to Invoice Matching tables:

Invc_head

Invc_non_merch

## Function Level Description

#### Init ()

1. Call retek_init to determine the max_commit_ctr.
2. Get the system date and time using **fetchSysdate**().
3. Call the function **fetchSaSystemOptions**() to retrieve the unit_of_work (UOW).

#### Process ()

1. Process() gets the store_days to be exported using the first driving cursor.
2. For each store_day to be exported, process() checks whether there are errors associated with this store_day (**fetchStoreDayErrorCount())**. If so, processing for this store_day is skipped and step 2 is repeated for the next store_day.
3. Process() then attempts to obtain a read lock for this store day. If unsuccessful, processing for this store_day is skipped and step 2 is repeated for the next store_day.
4. Call **processStoreDay()** to process the store day transactions for DSD invoices.
5. Call **postInvoices()** to create the appropriate invc_head and/or invc_non_merch records.
6. Call **processStoreDayTotals()** to process the escheatment totals
7. Call **markStoreDayExported()** – updates sa_export_log, sets status to exported
8. Call **release_lock()** to release the lock on the store/day.
9. Steps 2-7 are repeated until there are no more store_days.

#### Final ()

1. Clean up any memory used.
2. Call **updateStoreDayExported**() to update sa_export_log for the last store day and system code (SYSE_IM).
3. Call **retek_close()**
4. Call **retek_refresh_thread()**

#### processStoreDay()

1. This function uses the second driving cursor to retrieve all transactions associated with direct store delivery invoice transactions. This function should fetch the appropriate data for inserts into invc_head and invc_non_merch tables.
2. Calls **PostInvoices()** posts the invoices to invc_head and invc_non_merch tables.
3. Calls the library function **markTransactionExported()** to insert a record into sa_exported.

#### processStoreDayTotals()

This function will loop through the library function **getTotals()** for the current store day and SYSE_IM usage type.

The getTotals() function is being modified to include pass back ref_no1, ref_no2 and ref_no3. Ref_no1 will contain the partner_id, ref_no2 the partner_type and ref_no3 the escheat amount.

```
int getTotal(
   char       *is_store_day_seq_no,
   const char *is_usage_type,
   char       *os_total_seq_no,
   char       *os_total_value,
   char       *os_rev_no,
   char       *os_total_type
   char       *os_total_type_id,
   char    os_ref_no1,
   char    os_ref_no2,
   char    os_ref_no3
   char       *os_status,
   long        il_max_counter,
   long        il_multiplier);
```

1. Call **post_escheat_invoice()** to create the appropriate invc_head and/or invc_non_merch records.
2. Mark the total exported by calling the library function **markTotalExported()**.

#### calcDueDate()

Select duedays from the terms table for the terms associated with the supplier or partner on sa_tran_head.vendor_no. Due_date = invoice date + duedays. Invoice dates should be sa_tran_head.tran_datetime that was fetched in the first cursor. The terms discount percent should be taken from the terms table.

```
     SELECT TO_CHAR((TO_DATE(invoice date, 'YYYYMMDDHH24MISS') +
duedays),'YYYYMMDD'),
                    percent
         FROM terms
      WHERE terms = terms of the supplier or partner
```

#### getNewInvcId()

This function should call the package INVC_SQL.NEXT_INVC_ID to retrieve the next invoice id in the sequence. Include standard package error handling around the package call.

#### postInvoices()

Note * Partner, supplier, and date information will all be validated in ReSA so there is no need to validate in the export. ReSA validates to ensure that merchandise invoices are only associated with a supplier, not a partner. Validates partner, date, and numbers as well.

Calls **check_invc_exists()** to check if the invoice exists on the invc_head tables. If a record is found, update it, otherwise call **GetNewInvcId()** and insert a new record to invc_head. For non-merchandise invoices, be sure to update/insert the invc_non_merch record as well. A record should only be written to invc_non_merch if the invoice is a non-merchandise invoice. This can be determined by looking at the sub_tran_type on sa_tran_head. If the sub_tran_type is 'EV', then it's a non-merchandise invoice. If it's 'MV', then it's a merchandise invoice. If the invoice does not exist on the invc_head table call **post_new_invoice()**. If the invoice already exists on the invc_head table call **update_invoice().**

1. Call **calcDueDate()** to determine what the due date is for each invoice.
2. Call **get_invc_head_data()** to get other columns : terms,payment_method,freight_terms,currency_code,exchange_rate,addr_key
3. for the invoice associated with a supplier or partner.
4. Use the mapping below to determine what values to use when inserting into invc_head and/or invc_non_merch.

## Field Mapping between ReSA and Invoice Matching

| INVC_HEAD | Required? | Datatype | Sales Audit Value |
|---|---|---|---|
| INVC_ID | NOT NULL | NUMBER(10) | If the invoice is not matched with an existing one, invc_id will be a system generated number (invc_sql.next_invc_id). |
| INVC_TYPE | NOT NULL | VARCHAR2(1) | 'I' for Merchandise Invoice,<br>'N' for Non-merchandise Invoice |
| SUPPLIER | | NUMBER(10) | DSD - Sa_tran_head.vendor_no<br>Escheat – NULL |
| EXT_REF_NO | | VARCHAR2(30) | DSD - Sa_tran_head.vendor_invc_no<br>Escheat – 'E' concatinated with State or Partner ID (Which will be state) |
| STATUS | NOT NULL | VARCHAR2(1) | Code for the status of the invoice. Valid values are U for unmatched, R for partially matched, M for matched, A for approved and P for posted. Invoice statuses are held on the codes table under the code type 'IMST'.<br>DSD - Default to 'U' using IMST_U.<br>Escheat – Default to 'A' using IMST_A |
| EDI_INVC_IND | NOT NULL | VARCHAR2(1) | 'N' |
| EDI_SENT_IND | NOT NULL | VARCHAR2(1) | 'N' |

| INVC_HEAD | Required? | Datatype | Sales Audit Value |
|---|---|---|---|
| MATCH_FAIL_IND | NOT NULL | VARCHAR2(1) | Indicates whether or not an invoice has failed a match attempt. Valid values are 'Y' or 'N'. Default to 'N' using YSNO_N. |
| REF_INVC_ID | | NUMBER(10) | N/A – Used for types 'C', 'D', 'R'. Default to NULL. |
| REF_RTV_ORDER_NO | | NUMBER(6) | N/A – Used for types 'C', 'D', 'R'. Default to NULL. |
| REF_PRICE_CHANGE | | NUMBER(8) | N/A – Used for types 'C', 'D', 'R'. Default to NULL. |
| REF_RSN_CODE | | VARCHAR2(6) | N/A – Used for types 'C', 'D', 'R'. Default to NULL. |
| TERMS | | VARCHAR2(15) | Defaulted from sups or partner table if DSD transaction or partner table if escheatment. |
| DUE_DATE | NOT NULL | DATE | Defaulted based on terms of the supplier or partner. |
| PAYMENT_METHOD | | VARCHAR2(6) | Code identifying the payment method for the invoice, indicating how the invoice will be paid. Valid values include 'LC' for letter of credit, 'WT' for wire transfer, and 'OA' for open account. Other values maybe added by the client as desired.<br>Payment methods will be held on the codes table under a code type of 'PAYM'.<br>Defaulted based on the payment_method of the vendor on the invoice for Merchandise Invoice.<br>Default to NULL for Non merchandise Invoice. |
| TERMS_DSCNT_PCT | | NUMBER(12,4) | Discount that will be applied to the invoice if the invoice is paid by the due date. Default to the terms.pct of the terms associated with the vendor on the invoice. |
| TERMS_DSCNT_APPL_IND | NOT NULL | VARCHAR2(1) | Indicates whether or not the terms discount has been applied to the total cost of the invoice or not. Valid values are 'Y' or 'N'.<br>Default to 'N' using YSNO_N. |

| INVC_HEAD | Required? | Datatype | Sales Audit Value |
|---|---|---|---|
| TERMS_DSCNT_APPL_NON_MRCH_IND | NOT NULL | VARCHAR2(1) | This field will indicate if the specified terms discount should be applied to non-merchandise costs.<br>Default to 'N' using YSNO_N. |
| FREIGHT_TERMS | | VARCHAR2(2) | Indicator that references the freight terms associated with the invoice. Default from sups table for merchandise invoice.<br>Non Merchandise Invoice: NULL. |
| CREATE_ID | NOT NULL | VARCHAR2(30) | 'ReSA' |
| CREATE_DATE | NOT NULL | DATE | The data the invoice was entered in the system – vdate |
| INVC_DATE | NOT NULL | DATE | Date the invoice was issued by the supplier – sa_tran_head.tran_datetime for existing invoice.<br>For New Invoice default to vdate. |
| MATCH_ID | | VARCHAR2(30) | Oracle user ID of the user that matched the invoice. Default to NULL. |
| MATCH_DATE | | DATE | Date the invoice was matched. Default to NULL. |
| APPROVAL_ID | | VARCHAR2(30) | Oracle user ID of the user that approved the invoice match. Default to NULL. |
| APPROVAL_DATE | | DATE | Date the invoice match was approved. Default to NULL. |
| FORCE_PAY_IND | NOT NULL | VARCHAR2(1) | Indicates whether or not the invoice is to be force paid (paid before being matched to receipts). Valid values are Y or N.<br>Default to 'N' using YSNO_N. |
| FORCE_PAY_ID | | VARCHAR2(30) | Oracle ID of the user that marked the invoice for force payment. This field will only have a value if the force_pay_ind = 'Y'. Default to NULL. |
| POST_DATE | | DATE | Date the invoice was posted to the AP staging tables.<br>Default to NULL. |

| INVC_HEAD | Required? | Datatype | Sales Audit Value |
|---|---|---|---|
| CURRENCY_CODE | NOT NULL | VARCHAR2(3) | Code identifying the currency in which the invoice is held.<br>Default the supplier's or partner's currency. |
| EXCHANGE_RATE | | NUMBER(20,10) | Exchange rate at which the invoice is held. Default from currency_rates table based on the currency of the supplier/partner. |
| TOTAL_MERCH_COST | | NUMBER(20,4) | Total merchandise cost for the invoice. This field will be held in the invoice currency. For DSD merchandise invoices, this field should hold the total cost of the invoice (sa_tran_tender.tender_amt). Default to NULL for DSD non-merchandise and escheatment invoices. |
| TOTAL_QTY | | NUMBER(12,4) | Total quantity of items on the invoice. This field is optional, and only needs to be entered if total quantity matching will be performed on the invoice.<br>Quantity will not be captured in ReSA. Default to NULL. |
| DIRECT_IND | NOT NULL | VARCHAR2(1) | Indicates whether the invoice was created for a direct store delivery order via the Quick Order Entry form in which the invoice was already paid. Valid values are 'Y' –Yes and 'N' -No.<br>**Default to 'Y' using YSNO_Y.** |
| PARTNER_TYPE | | VARCHAR2(6) | Type of partner assigned to the invoice. This field will always be 'EV' for Expense Vendor. Default using PTAL_EV. |
| PARTNER_ID | | VARCHAR2(10) | DSD - Sa_tran_head.vendor_no<br>Escheatment – ref_no1 (partner_id)<br>Partner assigned to the invoice. Partners can be assigned to any invoice type except merchandise invoices. |

| INVC_HEAD | Required? | Datatype | Sales Audit Value |
|---|---|---|---|
| ADDR_KEY | NOT NULL | NUMBER(6) | Indicates which vendor invoice address should be associated with the invoice.<br>Default to the primary address for the invoice address type on ADDR table (Addr_type = 05 and primary_addr_ind = 'Y'). |
| PAID_IND | NOT NULL | VARCHAR2(1) | Default to 'Y' (YSNO_Y) for all DSD transactions –they will be assumed paid. Set to 'N' (YSNO_N) for Escheatment invoices – they will be assumed not paid . |
| PAYMENT_REF_NO | | VARCHAR2(13) | DSD - Sa_tran_head.payment_ref_no<br>Escheat – NULL |
| PAYMENT_DATE | | DATE | Date that the invoice was paid from the POS system – sa_tran_head.tran_datetime.<br>Escheat – NULL |
| PROOF_OF_DELIVERY_NO | | VARCHAR2(30) | DSD – Sa_tran_head.proof_of_delivery_no<br>Escheat – NULL |
| CE_ID | | NUMBER(10) | NULL |
| OBLIGATION_KEY | | NUMBER(10) | NULL |
| COMMENTS | | VARCHAR2(255) | NULL |

| INVC_NON_MERCH | Required? | Datatype | Sales Audit Value |
|---|---|---|---|
| INVC_ID | NOT NULL | NUMBER(10) | The invc_id on invc_non_merch should correspond to the one on invc_head. |
| NON_MERCH_CODE | NOT NULL | VARCHAR2(6) | Code identifying the non-merchandise cost being added to the invoice. These codes will be held on the non_merch_code_head table .<br>For DSD transactions, this field should be set to the reason_code from sa_tran_head. Should be 'E' for Escheatment invoices. |

| INVC_NON_MERCH | Required? | Datatype | Sales Audit Value |
|---|---|---|---|
| NON_MERCH_AMT | | NUMBER(20,4) | Amount of the non-merchandise cost, specified by the non-merchandise code that has been invoiced for. This field will be held in the invoice currency.<br>DSD – sa_tran_tender.tender_amt<br>Escheat – total_value from **getTotals()** |
| VAT_CODE | | VARCHAR2(6) | Default to NULL. |
| SERVICE_PERF_IND | NOT NULL | VARCHAR2(1) | Indicates if a service non-merchandise cost has actually been performed. Valid values are 'Y' (service has been performed) or 'N' (service has not been performed or non-merchandise cost is not a service cost).<br>For DSD, if proof of delivery is provided, this field should be 'Y' (YSNO_Y).<br>Should be set to 'N' (YSNO_N) for escheatment invoices. |
| STORE | | NUMBER(4) | Indicates the store at which the service was performed. Should be populated with the DSD invoice store number. Null for escheatment. |

## Scheduling Considerations

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

Scheduling Diagram: This module should be executed after the ReSA transaction import process. This module should not be run simultaneously with other modules: saexprms, saexprdw, saexpach, saexpuar, and saexpgl.

Threading Scheme: N/A

## Locking Strategy

Locking will be performed via the get_lock and release_lock library functions to lock the store-day during processing.

## Restart/Recovery

The logical unit of work for this module is defined as a unique store/day combination. Records will be fetched, updated and inserted based on the commit_max_ctr specified on the restart_control table. Only two commits will be done, one to establish the store/day lock and another at the end, to release the lock after a store/day has been completely processed.

In case of failure, we rollback all work done to the point right after the call to get_lock() and then we release the lock. Thus, we assume that the rollback segment is large enough to hold all inserts into sa_exported for one store_day. If this is not the case, we need to increase the size of the rollback segment. The EXEC SQL SAVEPOINT statement is used to save the state of the database after the call to get_lock().

Restart recovery is implicit in the program, as only store_days with a sa_export_log.status of 'R'eady (SAES_R) will be selected for processing. Since we set this status to 'E'xported (SAES_R) after a store_day is processed, then on restart, store_days that have been processed will be skipped.

## Driving Cursors

The program has three driving cursors: one to fetch store_days to be exported, another to fetch invoice matching transactions to be exported for the store_days fetched in the first cursor, and the last to fetch escheatment totals to be exported to Invoice Matching.

The following cursor will be used to retrieve the valid store/day identifiers that must be processed:

```
SELECT sd.store_day_seq_no,
               TO_CHAR(sd.business_date, 'YYYYMMDD'),
               el.seq_no,
                sd.store
    FROM sa_store_day sd, sa_export_log el
  WHERE sd.store_day_seq_no = el.store_day_seq_no
       AND sd.store_status = :SASS_C  /* Closed */
       AND sd.data_status = :SADS_F   /* Fully Loaded */
       AND sd.audit_status IN (:SAAS_A, :SAAS_S, :SAAS_H) / *Audit Pending, Store
Errors Pending, HQ Errors Pending */
       AND el.system_code = :SYSE_IM
       AND el.status = :SAES_R       /* Ready to be exported */
ORDER BY sd.store, sd.business_date;
```

The second driving cursor selects DSD invoice transactions from transaction tables ('PADIOU').

```
SELECT h.tran_seq_no,
       h.rev_no,
       TO_CHAR(h.tran_datetime,'YYYYMMDDHH24MISS'),
       h.tran_no,
       h.tran_type,
       h.sub_tran_type,
       h.reason_code,
       h.vendor_no,
       h.vendor_invc_no,
       h.payment_ref_no,
       h.proof_of_delivery_no,
       h.status,
       t.tender_amt
       :SAFD_P /* Positive Transaction/Total */
  FROM sa_tran_head h
 WHERE h.store_day_seq_no = :is_store_day_seq_no
    AND    h.tran_seq_no  = t.tran_seq_no
   AND h.tran_type IN (:TRAT_PAIDOU)
   AND h.status != :SAST_D
  AND h.sub_tran_type IN ('EV','MV')
   AND NOT EXISTS
       (SELECT ex.tran_seq_no
          FROM sa_exported ex
         WHERE ex.tran_seq_no = h.tran_seq_no
```

```
           AND ex.store_day_seq_no = :is_store_day_seq_no)
AND NOT EXISTS      /* and no errors for the transaction. */
           (SELECT er.tran_seq_no
              FROM sa_error er, sa_error_impact ei, sa_tran_head h
             WHERE h.tran_seq_no = er.tran_seq_no
               AND er.error_code = ei.error_code
               AND ei.system_code = :SYSE_IM
               AND er.hq_override_ind != :YSNO_Y))
ORDER BY h.tran_datetime;
```

The last driving cursor is embedded in the **getTotals()** function. This function is called with a usage type of SYSE_IM. For each escheatment transaction that is processed, write to the invc_head and invc_non_march tables.

# Sales Audit Export to RDW [saexprdw]

## Design Overview

The purpose of this batch module is to fetch all corrected sale and return transactions that do not have RDW errors from the Retek Sales Audit (ReSA) database tables for transmission to the Retek Merchandising SystemData Warehouse (RDW). The data will be sent at the store day level. If the transaction has a status of Deleted and it has previously been transmitted, a reversal of the transaction will be sent.

Four files of type RDWT, RDWF, RDWS and RDWC will be created for each store_day. See the file Interface File – SA to RDW.doc for more information.

RDW requires that the employee id be sent. saexprdw is expected to do this by mapping a cashier ID to an employee ID using the sa_store_emp table. However, the latter may not always be populated and thus, we send a blank field to RDW in this case.

Multi threading based on store was added to this program in version 11.0.X.

### Tables Affected:

| TABLES | SELECT | INSERT | UPDATE | DELETE |
|---|---|---|---|---|
| sa_store_day | Yes | No | No | No |
| sa_export_log | Yes | No | Yes | No |
| sa_error | Yes | No | No | No |
| sa_error_impact | Yes | No | No | No |
| sa_tran_head | Yes | No | No | No |
| sa_tran_item | Yes | No | No | No |
| sa_tran_disc | Yes | No | No | No |
| sa_tran_tender | Yes | No | No | No |
| sa_customer | Yes | No | No | No |
| sa_tran_head_rev | Yes | No | No | No |
| sa_tran_item_rev | Yes | No | No | No |
| sa_tran_disc_rev | Yes | No | No | No |
| sa_tran_tender_rev | Yes | No | No | No |
| sa_store_emp | Yes | No | No | No |
| Sasa_total | Yes | No | No | No |
| sa_exported | Yes | No | No | No |
| sa_exported_rev | Yes | No | No | No |
| sa_store_price_hist_temp | Yes | Yes | No | Yes |

## Global Variable Descriptions

| Gobal Variable | Description |
|---|---|
| pi_commit_max_ctr | Commit max counter used for array fetches. |
| ps_num_threads | Maximum number of threads |
| ps_thread_val | Thread value |
| pi_proc_cnt | Commit max counter used for array fetches. |
| pl_multiplier | Multipliers to remove decimals from numbers. |
| ps_sysdate | Current sysdate value from the database. |
| ps_store | Store ID from store/day driving cursor. |
| ps_business_date | Business date from store/day driving cursor. |
| ps_temp_rdwtfile | Temporary file name to be used for the RDWT file. |
| ps_temp_siftenderfilerdwffile | Temporary file name to be used for the RDWF file. |
| ps_temp_rdwsfile | Temporary file name to be used for the RDWS file. |
| ps_temp_rdwcfile | Temporary file name to be used for the RDWC file. |
| ps_TranHeadNo | Current transactions tran_seq_no. |
| pi_curtrat | Current transactions transaction type converted to an enum. |
| pi_tdetl_count | TDETL record count for TTAIL record in the RDWT file. |
| ps_total_sales_value | Total sales value of a TITEM record minus any discounts from associated IDISC records. |
| pl_rdwc_line_ctr | Line counter for the RDWC file. |
| pl_rdwf_line_ctr | Line counter for the RDWF file. |
| pl_rdws_line_ctr | Line counter for the RDWS file. |
| pl_rdwt_line_ctr | Line counter for the RDWT file. |
| SIFTenderFileRDWFFile | File pointer for the RDWF file. |
| RDWTFile | File pointer for the RDWT file. |
| RDWSFile | File pointer for the RDWS file. |
| RDWCFile | File pointer for the RDWC file. |
| pi_num_locks_not_released | Counter for the number of store/day locks that could not be released. |
| pi_num_non_fatal_errors | Counter for the number of non-fatal errors encountered:<br>Store/day lock could not be release.<br>An unexpected total was encountered.<br>Could not translate a cashier POS ID to an employee ID.<br>Could not translate a salesperson POS ID to an employee ID. |

## Function Level Description

**main()**

int argc

char *argv[]

Check command line for required arguments.

Call LLOGON to connect to the database.

Call Init to initialize the program.

Call process to export the available RDW data.

Report unlocking errors.

Report non-fatal errors.

Call final to cleanup.

**init()**

No arguments

This function initializes Restart recovery.

Get the value of sa_system_options.unit_of_work by calling the library function fetchSaSystemOptions.

Initialize Oracle Number functions by calling OraNumInit.

Get a temporary filenames to use for generating the output files. Store these names in ps_temp_rdwtfile, ps_temp_siftenderfilerdwffile, ps_temp_rdwsfile, and ps_temp_rdwcfile.

**process()**

No arguments

Picks a store/day to be processed by fetching using the first driving cursor. Save the store ID in ps_store and the date in ps_business_date.

Attempt to lock the store/day with a call to get_lock. If this fails, go on to the next store/day.

Open RDWTFile, RDWSFile, RDWCFile and RDWFile, using temporary names generated in init.

Set pl_rdwc_line_ctr, pl_rdwf_line_ctr, pl_rdws_line_ctr and pl_rdwt_line_ctr to 0.

Call fetchSysDate to get the current date/time. Store it in ps_sysdate.

Increment pl_file_counter.

Write records into sa_store_price_hist_temp table. Get latest tran_type for all items in a given store and write it to the temp table.

Call WrRDWFHead to write a RDWT FHEAD record to the RDWT file.

Call WrRDWFHead to write a RDWF FHEAD record to the RDWF file.

Call processStoreDay to process the store/days transactions.

Increment pl_rdwt_line_ctr.

Call WrRDWFTail to write a RDWT FTAIL record to the RDWT file.

Call WrRDWFTail to write a RDWF FTAIL record to the RDWF file.

Call processStoreDayTotals to process all totals for a given store day.

Update the status in sa_export_log to Complete by calling the library function markStoreDayExported.

Close the RDWTFile, SIFTenderFileRDWFFile, RDWSFile and RDWCFile and rename them appropriately (*file-type_store_business-date_current-datetime*).

Call to release_lock and go on to the next store/day. This function commits as a side effect, thus committing the changes to the database.

#### final()

int ii_process_ret

Remove the temporary file, if we failed to finish (ii_proces_ret is not OK).

Call retek_close.

Call trace_threading to clean up all internal processing

Call retek_refresh_thread.

#### processStoreDay()

char is_store_day_seq_no[NULL_BIG_SEQ_NO]

char is_store[NULL_LOC]

char ps_sysdate[NULL_DATETIME]

For each transaction from the store/day being processed, get the following information from the second driving cursor and call processTransHead with the information.

| Table | Column | Description |
|---|---|---|
| Sa_tran_head | Tran_seq_no | |
| Sa_tran_head | Rev_no | |
| Sa_tran_head | Tran_datetime | Format YYYYMMDDHH24MISS |
| Sa_tran_head | Register | |
| Sa_tran_head | Tran_no | |
| Sa_customer | Cust_id_type | via an outer join |
| Sa_customer | Cust_id | via an outer join |
| Sa_tran_head | Reason_code | |
| Sa_tran_head | Tran_type | |
| Sa_tran_head | Sub_tran_type | |
| Sa_tran_head | Orig_tran_no | |
| Sa_tran_head | Orig_reg_no | |
| Sa_tran_head | Ref_no1 | |
| Sa_tran_head | Ref_no2 | |
| Sa_tran_head | Ref_no3 | |
| Sa_tran_head | Ref_no4 | |
| Sa_tran_head | Vendor_no | |
| Sa_tran_head | Status | |
| Sa_tran_head | Value | 'SIGN_N' or 'SIGN_P' depending on the sign of value. |
| Sa_tran_head | Value | Absolute value multiplied by 10000. |

| Table | Column | Description |
|---|---|---|
| | Transaction Sign | 'SAFD_P' if the transaction has not been deleted (status != 'SAST_D') and there are no errors and it has not been exported.<br>'SAFD_N' if the transaction has been deleted (status = 'SAST_D') and it has been exported after being exported. |
| Sa_exported | Exp_datetime | Only for transactions with a Transaction Sign of 'SAFD_N'.<br>Format YYYYMMDDHH24MISS |
| Sa_store_emp | Emp_id | Pos_id = cashier |
| Sa_store_emp | Emp_id | Pos_id = salesperson |
| Sa_tran_head | Banner_no | |
| Sa_tran_head | Cust_order_no | Customer order number |
| Sa_tran_head | Cust_order_date | Format YYYYMMDD |

Copy the cashier and salesperson employee ID's to ps_last_cash_id and ps_last_sp_id.

Calls the library function **markTransactionExported** to insert a record into sa_exported for each transaction.

#### processTransHead()

char is_store_day_seq_no[NULL_BIG_SEQ_NO]

char is_store[NULL_STORE]

char is_day[NULL_DAY]

struct pt_sa_tran_head ir_sa_tran_head

If the transaction status is deleted (SAST_D) and it has been previously exported, then call retrieveTransHeadRev. Also, if the revision number of the transaction is not 1, then a previous revision may have been exported; call retrieveTransHeadRev to get the exported revision (for full disclosure purposes).

Call retrieveTransItem, retrieveTransDisc and retrieveTransTender to obtain the items, discounts and tenders for the transaction, both Positive transactions and Negative ones.

Call saveData for both the Positive and Negative transactions to write the information into memorythe RDW files.

The cust_id_type, cust_id, and emp_ids for cashier and salesperson have to be copied to global variables for future use in WrRDWTHead.

#### retrieveTransHeadRev()

char is_store_day_seq_no[NULL_BIG_SEQ_NO]

char is_store[NULL_STORE]

char is_day[NULL_DAY]

struct pt_sa_tran_head *or_sa_tran_head_rev

This function gets the sa_tran_head_rev record that needs to be processed. A record needs to be processed if it has been previously exported.

| Table | Column | Description |
|---|---|---|
| Sa_tran_head_rev | Tran_seq_no | |
| Sa_tran_head_rev | Rev_no | |
| Sa_tran_head_rev | Tran_datetime | Format YYYYMMDDHH24MISS |
| Sa_tran_head_rev | Register | |
| Sa_tran_head_rev | Tran_no | |
| Sa_store_empSa_tran_head_rev | Emp_idCashier | Pos_id = cashier via an outer join separate from salesperson |
| Sa_store_empSa_tran_head_rev | Emp_idSalesperson | Pos_id = salesperson via an outer join separate from cashier |
| Sa_customer | Cust_id_type | via an outer join |
| Sa_customer | Cust_id | via an outer join |
| Sa_tran_head_rev | Reason_code | |
| Sa_tran_head_rev | Tran_type | |
| Sa_tran_head_rev | Sub_tran_type | |
| Sa_tran_head_rev | Orig_tran_no | |
| Sa_tran_head_rev | Orig_reg_no | |
| Sa_tran_head_rev | Ref_no1 | |
| Sa_tran_head_rev | Ref_no2 | |
| Sa_tran_head_rev | Ref_no3 | |
| Sa_tran_head_rev | Ref_no4 | |
| Sa_tran_head_rev | Vendor_no | |
| Sa_tran_head_rev | Status | |
| Sa_tran_head_rev | Value | 'SIGN_N' or 'SIGN_P' depending on the sign of value. |
| Sa_tran_head_rev | Value | Absolute value multiplied by 10000. |
| | Transaction Sign | 'SAFD_N' |
| Sa_exported_rev | Exp_datetime | Only for transactions with a Transaction Sign of 'SAFD_N'.<br>Format YYYYMMDDHH24MISS |
| Sa_tran_head_rev | Banner_no | |
| Sa_tran_head_rev | Cust_order_no | Customer order number |
| Sa_tran_head_rev | Cust_order_date | Format YYYYMMDD |

If no data is found, than set or_sa_tran_head_rev->s_rev_no to –1.

#### retrieveTransItem()

char is_store_day_seq_no[NULL_BIG_SEQ_NO]

char is_store[NULL_STORE]

char is_day

char is_rev_no[NULL_SA_REV_NO]

long *ol_num_sa_tran_item

struct pt_sa_tran_item **or_sa_tran_item

This function gets all sa_tran_item records or sa_tran_item_rev (if is_rev_no is not –1) that need to be processed for a tran_seq_no.

| Table | Column | Description |
|---|---|---|
| Sa_tran_item | Tran_seq_no | |
| Sa_tran_item | Item_seq_no | |
| Sa_tran_item | Item_status | |
| Sa_tran_item | SkuItem | |
| Sa_tran_itemSa_tran_item | Ref_itemUpc | |
| Sa_tran_itemSa_tran_item | Non_merch_itemUpc_suppliment | |
| Sa_tran_item | Voucher_no | |
| Sa_tran_item | Dept | |
| Sa_tran_item | Class | |
| Sa_tran_item | Subclass | |
| Sa_tran_item | Standard_qty | 'SIGN_N' or 'SIGN_P' depending on the sign of qty. |
| Sa_tran_item | Standard_qty | Absolute value multiplied by 10000. |
| Sa_tran_item | Standard_unit_retail | 'SIGN_N' or 'SIGN_P' depending on the sign of unit_retail. |
| Sa_tran_item | Standard_unit_retail | Absolute value multiplied by 10000. |
| Sa_tran_item | Tax_ind | |
| Sa_tran_item<br>Sa_tran_item | Item_swiped_indItem_swipped_ind | |
| Sa_tran_item | Standard_orig_unit_retail | 'SIGN_N' or 'SIGN_P' depending on the sign of orig_unit_retail. |
| Sa_tran_item | Standard_orig_unit_retail | Absolute value multiplied by 10000. |
| Sa_tran_item | Item_type | |
| Sa_tran_item | Override_reason | |
| Sa_store_emp | Emp_id | |
| Sa_tran_item | Return_reason_code | |
| Sa_tran_item | Drop_ship_ind | |
| Sa_tran_item | Selling_item | |
| Sa_tran_item | Customer_order_line_no | |
| Sa_tran_item | Media_id | |
| Sa_store_price_hist_temp | Retail_type | |

The same columns as above are select from the sa_tran_item_rev table if the rev_no passed in is not –1.

Set *ol_num_sa_tran_item to the total number of records fetched.

**retrieveTransDisc()**

char is_store_day_seq_no[NULL_BIG_SEQ_NO]

char is_store[NULL_STORE]

char is_day[NULL_DAY]

char is_rev_no[NULL_SA_REV_NO]

long *ol_num_sa_tran_disc

struct pt_sa_tran_disc **or_sa_tran_disc

This function gets all sa_tran_disc or sa_tran_disc_rev records (if is_rev_no is not –1) for a tran_seq_no that needs to be processed.

| Table | Column | Description |
|---|---|---|
| Sa_tran_disc | Tran_seq_no | |
| Sa_tran_disc | Item_seq_no | |
| Sa_tran_disc | Discount_seq_no | |
| Sa_tran_disc | Rms_promoDiscount_type | |
| Sa_tran_disc | Promotion | |
| Sa_tran_disc | Discount_type | |
| Sa_tran_disc | Coupon_no | |
| Sa_tran_disc | Coupon_ref_noCoupon_ref_no | |
| Sa_tran_disc | Standard_qty | 'SIGN_N' or 'SIGN_P' depending on the sign of qty. |
| Sa_tran_disc | Standard_qty | Absolute value multiplied by 10000. |
| Sa_tran_disc<br>Sa_tran_item | (Unit_retail * standard_qty) – (unit_discount_amt * qty) | Absolute value multiplied by 10000. |
| Sa_tran_disc<br>Sa_tran_item | (Unit_retail * standard_qty) – (unit_discount_amt * qty) | 'SIGN_N' or 'SIGN_P' depending on the sign of the expression. |
| Sa_tran_disc | Standard_unit_discount_amt | 'SIGN_N' or 'SIGN_P' depending on the sign of unit_discount_amt. |
| Sa_tran_disc | Standard_unit_discount_amt | Absolute value multiplied by 10000. |
| Sa_tran_disc | | |

The same columns as above are select from the sa_tran_disc_rev table if the rev_no passed in is not –1.

Set *ol_num_sa_tran_disc to the total number of records fetched.

**retrieveTransTender()**

char is_store_day_seq_no[NULL_BIG_SEQ_NO]

char is_store[NULL_STORE]

char is_day[NULL_DAY]

char is_rev_no[NULL_SA_REV_NO]

long *ol_num_sa_tran_tender

struct pt_sa_tran_tender **or_sa_tran_tender

This function gets all sa_tran_tender or sa_tran_tender_rev records (if is_rev_no is not –1) for a tran_seq_no that needs to be processed.

| Table | Column | Description |
|---|---|---|
| Sa_tran_tender | Tran_seq_no | |
| Sa_tran_tender | Tender_seq_no | |
| Sa_tran_tender | Tender_type_group | |
| Sa_tran_tender | Tender_type_id | |
| Sa_tran_tender | Tender_amt | 'SIGN_N' or 'SIGN_P' depending on the sign of tender_amt. |
| Sa_tran_tender | Tender_amt | Absolute value multiplied by 10000. |
| Sa_tran_tender | Cc_no | |
| Sa_tran_tender | Cc_auth_no | |
| Sa_tran_tender | Cc_auth_src | |
| Sa_tran_tender | Cc_cardholder_verf | |
| Sa_tran_tender | Cc_exp_date | Format YYYYMMDD |
| Sa_tran_tender | Cc_entry_mode | |
| Sa_tran_tender | Cc_term_id | |
| Sa_tran_tender | Cc_spec_cond | |
| Sa_tran_tender | Voucher_no | |
| Sa_tran_head<br>Sa_voucher | Business_date – iss_date | Voucher age |
| Sa_voucher | Escheat_date | |
| Sa_tran_tender | Coupon_no | |
| Sa_tran_tender | Coupon_ref_no | |

The same columns as above are select from the sa_tran_tender_rev table if the rev_no passed in is not –1.

Set *ol_num_sa_tran_tender to the total number of records fetched.

**saveData()**

struct pt_sa_tran_head ir_sa_tran_head

long il_num_sa_tran_item

struct pt_sa_tran_item *ia_sa_tran_item

long il_num_sa_tran_disc

struct pt_sa_tran_disc *ia_sa_tran_disc

long il_num_sa_tran_tender

struct pt_sa_tran_tender *ia_sa_tran_tender

Creates RTLOG buffers for each transaction.

Set pi_curtrat to the current transaction type by calling trat_lookup.

Call ProcRecord WrRDWTHead to process the THEAD buffercurrent ia_sa_tran_head record if the transaction type (pi_curtrat) is TRATTT_COND, TRATTT_PAIDIN or TRATTT_PAIDOU.

For each item record:

Blank pad NULL values so we do not get all zeros in the VRTLOG.

Call tsv_lookahead to calculate the total sales value for later use.

Call ProcRecord WrRDWTHead to process the TITEM buffercurrent ia_sa_tran_item record.

For each item's discount record:

Call ProcRecord WrRDWTDetl to process the IDISC buffercurrent ia_sa_tran_disc record.

Increment ll_cur_sa_tran_disc.

For each tender record:

Call WRITE_TTEND to create a TTEND buffer.

Call ProcRecord WrRDWFDetl to process the TTEND buffercurrent ia_sa_tran_tender.

Increment cur_sa_tran_tender.

#### ProcessStoreDayTotals()

char is_store_day_seq_no[NULL_BIG_SEQ_NO]

char is_store[NULL_STORE]

char is_day[NULL_DAY]

const char is_usage_type[NULL_CODE]

This function will loop through the library function getBalTotals for the current store day.

Call WrRDWFHead to write this header to the RDWS file.

Increment pl_rdwc_line_ctr.

Call WrRDWFHead to write this header to the RDWC file.

For each total returned:

1. If the total_id is "OVRSHT_B", call **WrRDWCTDetl** then write the data to the RDWC file.
2. Else, if the cashier_id and the register_id are both nulls, call **WrRDWSTDetl** then write to the RDWS file.
3. Else, mark this as an error, since the RDWS file can only handle store level totals.
4. If the total is not a 'N'egative total, mark the total exported by calling the library function **markTotalExported**.

Increment pl_rdws_line_ctr.

Call **WrRDWFTail** to write this header to the RDWS file.

Increment pl_rdwc_line_ctr.

Call **WrRDWFTail** to write this header to the RDWC file.

#### WrRDWFHead()

char *is_file_type

FILE *is_file

long *iol_line_ctr

Set *iol_line_ctr to 1L. This is the appropriate global line counter variable for the file type.

Writes an RDW_FHEAD record (as defined in salib.h) to the specified output file. This must match the definition of the record in Interface File – SA to RDW.doc.

| Field | Type | Size | Source |
|---|---|---|---|
| Frecdesc | char | RDW_FRECDESC_SIZE | RDW_FHEAD_FRECDESC |
| Flineid | char | LEN_FILE_LINE_NO | *iol_line_ctr |
| file_type_definition | char | LEN_FILE_TYPE_DEF | is_file_type |
| file_create_date | char | LEN_DATETIME | p->file_create_dateps_sysdate |

Call **putrec** to write the record out to the RDWT or RDWF file.

#### WrRDWTHead()

RTL_TITEMpt_sa_tran_head *pir_head

Pt_sa_tran_item *ir_item

Set pi_tdetl_count to 0.

Increment pl_rdwt_line_ctr.

This function writes a RDW_THEAD record (as defined in salib.h) to the output file. This must match the definition of the record in Interface File – SA to RDW.doc.

If currently in a transaction block (pi_inTranBlock) than write it out by calling **WrRDWTTail**.

Increment pl_rdwt_line_ctr.

| Field | Type | Size | Source |
|---|---|---|---|
| Fredesc | char | RDW_FRECDESC_SIZE | RDW_THEAD_FRECDESC |
| Flineid | char | FT_NUMBER | pl_rdwt_line_ctr |
| Business_date | Date | FT_DATE | ps_business_date |
| tran_datetime | char | FT_DATE | ir_head->s_tran_datetime |
| Location | char | FT_NUMBER | RTLFHead.locationps_store |
| register_id | char | FT_VARCHAR | ir_head->s_register |
| cashier_id | char | FT_VARCHAR | ir_head->s_cashier |
| Salesperson_id | char | FT_VARCHAR | ir_item->s_salesperson<br>if NULL then use ir_head->s_salesperson |
| cust_id_type | char | FT_VARCHAR | ir_head->s_cust_id_type |
| cust_id_number | char | FT_VARCHAR | ir_head->s_cust_id |
| tran_no | char | FT_NUMBER | ir_head->s_tran_no |
| Orig_register | Char | FT_VARCHAR | ir_head->s_orig_reg_no |
| Orig_tran_no | Char | FT_NUMBER | Ir_head-> s_orig_tran_no |
| tran_seq_no | char | FT_VARCHAR | ir_head->s_tran_seq_no, |
| rev_no | char | FT_NUMBER | ir_head->s_rev_no |

| Field | Type | Size | Source |
|---|---|---|---|
| tran_sign | char | FT_VARCHAR | ir_head->s_tran_sign |
| tran_type | char | FT_VARCHAR | ir_head->s_tran_type |
| tran_type | char | FT_VARCHAR | TRAT_SALE |
| tran_type | char | FT_VARCHAR | TRAT_RETURN |
| tran_type | char | FT_VARCHAR | TRAT_VOID |
| sub_tran_type | char | FT_VARCHAR | ir_head->s_sub_tran_type<br>If NULL then use - 1 |
| emp_cashier_no | char | FT_VARCHAR | ir_head->s_ref_no1 if sub_tran_type = TRAS_EMP<br>else use -1 |
| receipt_ind | char | FT_VARCHAR | ir_head->s_ref_no1<br>if tran_type = TRAT_RETURN |
| reason_code | char | FT_VARCHAR | ir_head->s_reason_code<br>if NULL use -1 |
| vendor_no | char | FT_VARCHAR | ir_head->s_vendor_no<br>if tran_type = TRAT_PAIDOU |
| item_type | char | FT_VARCHAR | ir_item->s_item_type if ir_item->s_item_type is either SAIT_ITEM or SAIT_REF.<br>SAIT_GCN if ir_item->s_item_type is SAIT_GCN. |
| item_no | char | FT_VARCHAR | ir_item->s_item if ir_item->s_item_type is SAIT_ITEM.<br>ir_item->s_voucher_no if ir_item->s_item_type is SAIT_GCN. |
| tax_ind | char | FT_VARCHAR | ir_item->s_tax_ind |
| drop_ship_ind | char | FT_VARCHAR | ir_item->s_drop_ship_ind |
| item_swiped_ind | char | FT_VARCHAR | ir_item->s_item_swiped_ind |
| Dept | char | FT_NUMBER | ir_item->s_dept |
| Class | char | FT_NUMBER | ir_item->s_class |
| Subclass | char | FT_NUMBER | ir_item->s_subclass |
| total_sales_qty | char | FT_NUMBER | ir_item->s_qty |
| total_sales_value | char | FT_NUMBER | ps_total_sales_value |
| override_reason | char | FT_VARCHAR | ir_item->s_override_reason if ir_item->s_override_reason is NULL, else use -1 |

| Field | Type | Size | Source |
|---|---|---|---|
| Return_reason_code | Char | FT_VARCHAR | ir_item->s_return_reason_code if ir_item->s_return_reason_code is NULL, else use - 1 |
| total_orig_sign | char | FT_VARCHAR | ir_itemp->s_qty_sign |
| total_sales_value | char | FT_NUMBER | ir_head->s_value |
| Weather | char | FT_VARCHAR | ir_head->s_ref_no1 if tran_type is TRAT_COND |
| Temperature | char | FT_VARCHAR | ir_head->s_ref_no2 if tran_type is TRAT_COND |
| Traffic | char | FT_VARCHAR | ir_head->s_ref_no3 if tran_type is TRAT_COND |
| Construction | char | FT_VARCHAR | ir_head->s_ref_no4 if tran_type is TRAT_COND |
| banner_id | char | FT_VARCHAR | ir_head->s_banner_id if ir_head->s_banner_id is NULL else use - 1 |
| Media_id | char | FT_VARCHAR | ir_item->s_media_id if ir_item!=NULL && strcmp(ir_item->s_media_id is NULL else use - 1 |
| customer_order_no | char | FT_VARCHAR | ir_head->s_customer_order_no if ir_head->s_customer_order_no is NULL else use -1 |
| customer_order_date | char | FT_VARCHAR | ir_head->s_customer_order_date |
| selling_item | char | FT_VARCHAR | ir_item->s_selling_item if ir_item!=NULL && strcmp(ir_item->s_selling_item is NULL else use - 1 |
| customer_order_line_no | char | FT_VARCHAR | ir_item->s_customer_order_line_no if ir_item!=NULL && strcmp(ir_item->s_customer_order_line_no is NULL else, use - 1 |

Call **putrec** to write the record out to the RDWT file.

#### WrRDWTDetl()

pt_sa_tran_head *ir_head

RTL_IDISCps_sa_tran_disc *pir_disc

Increment both pl_rdwt_line_ctr and pl_tdetl_count.

Writes an RDW_TDETL record (as defined in salib.h) to the RDWT output file. This must match the definition of the record in Interface File – SA to RDW.doc.

Increment both pl_rdwt_line_ctr and pl_tdetl_count.

| Field | Type | Size | Source |
|---|---|---|---|
| frecdesc | char | RDW_FRECDESC_SIZE | RDW_TDETL_FRECDESC |
| flineid | char | FT_NUMBER | ls_file_line_no |
| Discount_type | Char | FT_VARCHAR | ir_disc->s_disc_type |
| promo_tran_type | char | FT_VARCHAR | ir_disc->s_rms_promo_type |
| promo_no | char | FT_VARCHAR | ir_disc->s_promotion |
| Promo_comp | char | FT_VARCHAR | ir_disc->s_promo_comp |
| Coupon_no | Char | FT_VARCHAR | ir_disc->s_coupon_no |
| Coupon_ref_no | Char | FT_VARCHAR | ir_disc->s_coupon_ref_no |
| sales_qty | char | FT_NUMBER | ir_disc->s_qty |
| sales_sign | char | FT_VARCHAR | ir_disc->s_qty_sign |
| sales_value | char | FT_NUMBER | ps_total_sales_value |
| disc_value | char | FT_NUMBER | ir_discp-> s_unit_disc_amt |

Call **putrec** to write the record out to the RDWT file.

#### WrRDWTTail()

No arguments

Writes an RDW_TTAIL record (as defined in salib.h) to the RDWT output file. This must match the definition of the record in Interface File – SA to RDW.doc.

Increment pl_rdwt_line_ctr.

Set pi_inTranBlock to FALSE.

| Field | Type | Size | Source |
|---|---|---|---|
| frecdesc | char | RDW_FRECDESC_SIZE | RDW_TTAIL_FRECDESC |
| flineid | char | FT_NUMBER | pl_rdwt_line_ctr |
| tran_rec_counter | char | LEN_DTL_LINE_CNT | pi_tdetl_count |

Call **putrec** to write the record out to the RDWT file.

#### WrRDWFTail()

FILE *is_file

long *iol_line_ctr

Increments *iol_line_ctr. This is the appropriate global line counter variable for the file type.

Writes an RDW_FTAIL record (as defined in salib.h) to the specified output file. This must match the definition of the record in Interface File – SA to RDW.doc.

| Field | Type | Size | Source |
|---|---|---|---|
| frecdesc | char | RDW_FRECDESC_SIZE | RDW_FTAIL_FRECDESC |
| flineid | char | FT_NUMBER | *iol_line_ctr |
| file_rec_counter | char | FT_NUMBER | *iol_line_ctr – 2 |

Call **putrec** to write the record out to the RDWT or RDWF file.

#### WrRDWFDetl()

struct pt_sa_tran_head *ir_head,

struct pt_sa_tran_tender *ir_tend

Increment pl_rdwf_line_ctr.

| field | Type | Size | Source |
|---|---|---|---|
| frecdesc | char | RDW_FRECDESC_SIZE | RDW_FDETL_FRECDESC |
| flineid | char | FT_NUMBER | pl_rdws_line_ctr |
| business_date | char | FT_DATE | ps_business_date |
| tran_date | char | FT_DATE | ir_head->s_tran_datetime |
| location | char | FT_NUMBER | ps_store |
| cashier_id | char | FT_VARCHAR | ir_head->s_cashier if ir_head->s_cashier is NULL else use - 1 |

| field | Type | Size | Source |
|---|---|---|---|
| register_id | char | FT_VARCHAR | ir_head->s_register if ir_head->s_register is NULL else, use -1 |
| tran_sign | char | FT_VARCHAR | ir_head->s_tran_sign |
| Tran_seq_no | char | FT_VARCHAR | ir_head->s_tran_seq_no |
| Rev_no | | | ir_head->s_rev_no |
| Tran_type | | FT_VARCHAR | ir_head->s_tran_type |
| Tender_type_group | | FT_VARCHAR | ir_tend->s_tender_type_group |
| tender_type_id | | FT_VARCHAR | ir_tend->s_tender_type_id |
| tender_amt | | FT_VARCHAR | ir_tend->s_tender_amt |
| cc_no | | FT_VARCHAR | ir_tend->s_cc_no |
| cc_exp_date | | | ir_tend->s_cc_exp_date |
| cc_auth_no | | FT_VARCHAR | ir_tend->s_cc_auth_no |
| cc_auth_src | | FT_VARCHAR | ir_tend->s_cc_auth_src |
| cc_entry_mode | | FT_VARCHAR | ir_tend->s_cc_entry_mode |
| c_cardholder_verf | | FT_VARCHAR | ir_tend->s_cc_cardholder_verf |
| cc_terminal_id | | FT_VARCHAR | ir_tend->s_cc_terminal_id |
| cc_special_cond | | FT_VARCHAR | ir_tend->s_cc_special_cond |
| voucher_no | | FT_VARCHAR | ir_tend->s_voucher_no |
| voucher_age | | FT_VARCHAR | ir_tend->s_voucher_age |
| escheat_date | | FT_VARCHAR | ir_tend->s_escheat_date |
| coupon_no | | FT_VARCHAR | ir_tend->s_coupon_no |
| coupon_ref_no | | FT_VARCHAR | ir_tend->s_coupon_no |

Call **putrec** to write the record out to the RDWT or RDWF file.

#### WrRDWSTDetl()

char *is_status

char *is_total_id

char *is_ref_no1

char *is_ref_no2

char *is_ref_no3

char *is_total_value

Increment pl_rdws_line_ctr.

Writes an RDWS_TDETL record (as defined in salib.h) to the RDWS output file. This must match the definition of the record in Interface File – SA to RDW.doc.

Increment pl_rdws_line_ctr.

| Field | Type | Size | Source |
|---|---|---|---|
| frecdesc | char | RDW_FRECDESC_SIZE | RDW_FDETL_FRECDESC |
| flineid | char | FT_NUMBER | ls_file_line_no |
| tran_date | char | FT_DATE | ps_business_date |
| location | char | FT_NUMBER | ps_store |
| sales_sign | char | FT_VARCHAR | is_status |
| total_id | char | FT_VARCHAR | is_total_id |
| Ref_no1 | char | FT_VARCHAR | Is_ref_no1 |
| Ref_no2 | char | FT_VARCHAR | Is_ref_no2 |
| Ref_no3 | char | FT_VARCHAR | Is_ref_no3 |
| total_sign | char | FT_VARCHAR | SIGN_N or SIGN_P depending on whether or not is_total_value is negative. |
| total_amount | char | FT_NUMBER | Absolute value of is_total_value. |

Call **putrec** to write the record out to the RDWT file.

#### WrRDWCTDetl()

char *is_cashier_id
char *is_register_id
char *is_status
char *is_total_id
char *is_ref_no1
char *is_ref_no2
char *is_ref_no3
char *is_total_value
Increment pl_rdwc_line_ctr.
Writes an RDWC_FDETL record (as defined in salib.h) to the RDWC output file. This must match the definition of the record in Interface File – SA to RDW.doc.
Increment pl_rdwc_line_ctr.

| Field | Type | Size | Source |
|---|---|---|---|
| frecdesc | char | RDW_FRECDESC_SIZE | RDW_FDETL_FRECDESC |
| flineid | char | FT_NUMBER | ls_file_line_no |
| tran_date | char | FT_DATE | RTLFHead.ps_business_date |
| location | char | FT_NUMBER | RTLFHead.location ps_store |
| cashier_id | char | FT_VARCHAR | is_cashier_id if is_cashier_id is NULL else use - 1 |
| register_id | char | FT_VARCHAR | is_register_id if is_register_id is NULl else use -1 |
| sales_sign | char | FT_VARCHAR | is_status |

| Field | Type | Size | Source |
|---|---|---|---|
| total_id | char | FT_VARCHAR | is_total_id |
| Ref_no1 | char | FT_VARCHAR | Is_ref_no1 |
| Ref_no2 | char | FT_VARCHAR | Is_ref_no1 |
| Ref_no3 | char | FT_VARCHAR | Is_ref_no1 |
| total_sign | char | FT_NUMBER | SIGN_N or SIGN_P depending on whether or not is_total_value is negative. |
| total_amount | char | FT_NUMBER | Absolute value of is_total_value. |

Call **putrec** to write the record out to the RDWC file.

#### tsv_lookahead()

int i

This function calculates the total sales value (ps_total_sales_value) by "looking ahead" and summing up the item values and discounts for the current item record (i).

#### Blank_field()

char *is_field

int ii_len

This function fills the character array with spaces up to ii_len

#### Log_and_exit()

No arguments

This function logs message, calls the **final** function and exists with code 1.

## Stored Procedures / Shared Modules (Maintainability)

| Shared Module | Module Description |
|---|---|
| libretek.a functions | Refer to Library Design – retek.doc for details. |
| retek_init | Initialize restart recovery. |
| retek_close | Close restart recovery functions. |
| Retek_refresh_thread | Refresh the current thread so that it may be used again. |
| | |
| Libresa.a functions: | Refer to Library Design – ReSA.doc for details. |
| get_lock | used to establish a read lock on a store/day. |
| release_lock | used to release a store/day lock. |
| fetchSaSystemOptions | Fetch the values from the sa_system_options table. |
| fetchSysDate | Fetch the current SYSDATE value. |
| fetchStoreDayErrorCount | Fetch the number of errors that corresponds to a particular store/day and system. |
| markStoreDayExported | Mark a particular store/day and system as exported |
| markTransactionExported | Mark a particular transaction and system as exported. |

| Shared Module | Module Description |
|---|---|
| OraNum functions (Add, Sub, Mul, Div) | Used to perform arithmetic operations on strings containing large numbers. |
| getBalTotal | Get the specified balance totals. |
| putrec | Writes a record to a file. |

## Output Files

Data is output in the RDW file format. This is described in the file Interface File – SA to RDW.doc.

The filename convention for these valid RDWT, SIF Tender, RDWS and RDWC files will be rdwt_*store_businessdate_curdatetime*, rdwf_*store_businessdate_curdatetime*, rdws_*store_businessdate_curdatetime* and rdwt_*store_businessdate_curdatetime*. The files should start out with a temporary name generated by the Unix tempnam (3S) call and then be renamed with Unix rename (2) call when the files are complete.

## Scheduling Considerations

Processing Cycle: Anytime – Sales Audit 3.0 is a 24/7 system.

Scheduling Diagram: This will be run after auditors have made corrections to the data. This module should not be run simultaneously with other modules: saexprms, saexpim, saexpuar, saexpach, and saexpgl.

### Pre-Processing:

sagetref.pc to get waste data, and saimptlog.pc and saimptlogfin.pc to get post-void data.

### Post-Processing:

- stsldmat.pc (Sales Transaction SKU-Loc-Day-Minute ATomic) should be run to import data from the RDWT file into the RDW system.
- ttldmat.pc (Transaction Tender Loc-Day-Minute ATomic) should be run to import data from the RDWF file into the RDW system.
- lptotcldat.pc (Loss Prevention Totals Cashier-Loc-Day ATomic) should be run to import data from the RDWC file into the RDW system.
- lptotldat.pc (Loss Prevention Totals Loc-Day ATomic) should be run to import data from the RDWS file into the RDW system.

Threading Scheme: v_restart_store

## Locking Strategy

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

## Restart / Recovery

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

RDWS and RDWC formatted output files will be created with temporary names and renamed just before the end of store/day commit.

In case of failure, we rollback all work done to the point right after the call to **get_lock** and then we release the lock. Thus, we assume that the rollback segment is large enough to hold all inserts into sa_exported for one store_day. If this is not the case, we need to increase the size of the rollback segment. The EXEC SQL SAVEPOINT statement is used to save the state of the database after the call to **get_lock**.

There are 3 driving cursors in this module. The first picks a store/day to work on:

```
c_store_day CURSOR FOR
SELECT
       sd.store_day_seq_no,
       el.seq_no,
       sd.store,
       TO_CHAR(sd.business_date, 'YYYYMMDD'),
       ROWIDTOCHAR(el.rowid)
  FROM sa_store_day sd, sa_export_log el, v_restart_store vrs
 WHERE sd.store_day_seq_no = el.store_day_seq_no
   AND sd.store_status = :SASS_C    /* Closed                */
   AND sd.data_status  = :SADS_F    /* Fully loaded          */
   AND sd.audit_status = :SAAS_A    /* Audited, but no Errors */
   AND el.system_code  = :SYSE_RDW
   AND el.status       = :SAES_R    /* 'R'eady to be exported */
   AND vrs.num_threads = TO_NUMBER(:ps_num_threads)

   AND vrs.thread_val = TO_NUMBER(:ps_thread_val)
   AND vrs.driver_value = sd.store
      ORDER BY sd.store_day_seq_no, sd.store, sd.business_date;
```

Since RDW cannot accept data from a store_day with errors pending, we select store_days that have audit_status 'A' only. The library function **fetchStoreDayToBeExported** cannot be used here because it fetches store_days with an audit_status of 'E' (Errors pending).

The second driving cursor fetches the store/day transaction data to be output:

```
      c_tran_head CURSOR FOR
      SELECT h.tran_seq_no,
             h.rev_no,
             TO_CHAR( h.tran_datetime, 'YYYYMMDDHH24MISS'),
             NVL( h.register, ' '),
             NVL( TO_CHAR( h.tran_no), ' '),
             NVL( em.emp_id, ' '),
             NVL( em2.emp_id, ' '),
             NVL( c.cust_id_type, ' '),
             NVL( c.cust_id, ' '),
             NVL( h.reason_code, ' '),
             h.tran_type,
             NVL( h.sub_tran_type, ' '),
             NVL( TO_CHAR( h.orig_tran_no), ' '),
             NVL( h.orig_reg_no, ' '),
             NVL( h.ref_no1, ' '),
             NVL( h.ref_no2, ' '),
             NVL( h.ref_no3, ' '),
             NVL( h.ref_no4, ' '),
             NVL( h.vendor_no, ' '),
             h.status,
             DECODE( SIGN( h.value), -1, :SIGN_N, :SIGN_P),
             NVL( TO_CHAR( ABS(h.value) * :pl_multiplier), '0'),
             :SAFD_P,
             ' '

        ,
       NVL(to_char(h.banner_no), ' '),
```

```
          NVL(h.cust_order_no,' '),
          NVL(to_char(h.cust_order_date, 'YYYYMMDD'), ' ')
          FROM sa_tran_head h,
               sa_customer c,
               sa_store_emp em,
               sa_store_emp em2
         WHERE h.store_day_seq_no = TO_NUMBER(:is_store_day_seq_no)
           AND h.store            = TO_NUMBER(:is_store)
           AND h.day              = TO_NUMBER(:is_day)
           AND em.pos_id(+)       = h.cashier
           AND em.store(+)        = h.store
           AND em2.pos_id(+)      = h.salesperson
           AND em2.store(+)       = h.store
           AND h.tran_seq_no      = c.tran_seq_no(+)
           AND h.store            = c.store(+)
           AND h.day              = c.day(+)
           AND h.tran_type IN (:TRAT_SALE,   :TRAT_RETURN, :TRAT_EEXCH,
                               :TRAT_PAIDIN, :TRAT_PAIDOU, :TRAT_NOSALE,
                               :TRAT_VOID,   :TRAT_PVOID,  :TRAT_COND)
           AND (h.status = :SAST_P
                AND NOT EXISTS                  /* and no errors for the
transaction. */
                    (SELECT er.tran_seq_no
                       FROM sa_error er, sa_error_impact ei
                      WHERE h.tran_seq_no = er.tran_seq_no
                        AND h.store = er.store
                        AND h.day = er.day
                        AND er.error_code = ei.error_code
                        AND ei.system_code = :SYSE_RDW
                        AND er.hq_override_ind != :YSNO_Y))
           AND NOT EXISTS
              (SELECT e.store_day_seq_no
                 FROM sa_exported e
                WHERE h.store_day_seq_no = e.store_day_seq_no
                  AND h.store = e.store
                  AND h.day = e.day
                  AND h.tran_seq_no = e.tran_seq_no
                  AND e.system_code = :SYSE_RDW)
   UNION ALL
      SELECT h.tran_seq_no,
             h.rev_no,
             TO_CHAR( h.tran_datetime, 'YYYYMMDDHH24MISS'),
             NVL( h.register, ' '),
             NVL( TO_CHAR( h.tran_no), ' '),
             NVL( em.emp_id, ' '),
             NVL( em2.emp_id, ' '),
             NVL( c.cust_id_type, ' '),
             NVL( c.cust_id, ' '),
             NVL( h.reason_code, ' '),
             h.tran_type,
             NVL( h.sub_tran_type, ' '),
             NVL( TO_CHAR( h.orig_tran_no), ' '),
             NVL( h.orig_reg_no, ' '),
             NVL( h.ref_no1, ' '),
             NVL( h.ref_no2, ' '),
             NVL( h.ref_no3, ' '),
             NVL( h.ref_no4, ' '),
             NVL( h.vendor_no, ' '),
             h.status,
             DECODE( SIGN( h.value), -1, :SIGN_N, :SIGN_P),
             NVL( TO_CHAR( ABS(h.value) * :pl_multiplier), '0'),
             :SAFD_N,
             NVL( TO_CHAR( e.exp_datetime, 'YYYYMMDDHH24MISS'), ' ')
```

```
      ,
      NVL(to_char(h.banner_no), ' '),
      NVL(h.cust_order_no,' '),
      NVL(to_char(h.cust_order_date, 'YYYYMMDD'), ' ')
     FROM sa_tran_head h,
          sa_exported e,
          sa_customer c,
          sa_store_emp em,
          sa_store_emp em2
    WHERE h.store_day_seq_no = TO_NUMBER(:is_store_day_seq_no)
      AND h.store = TO_NUMBER(:is_store)
      AND h.day   = TO_NUMBER(:is_day)
      AND em.pos_id(+)  = h.cashier
      AND em.store(+)   = h.store
      AND em2.pos_id(+) = h.cashier
      AND em2.store(+)  = h.store
      AND h.tran_seq_no = c.tran_seq_no(+)
      AND h.store = c.store (+)
      AND h.day = c.day (+)
      AND h.tran_type IN (:TRAT_SALE,   :TRAT_RETURN, :TRAT_EEXCH,
                          :TRAT_PAIDIN, :TRAT_PAIDOU, :TRAT_NOSALE,
                          :TRAT_VOID,   :TRAT_PVOID,  :TRAT_COND)
      AND h.status in (:SAST_V, :SAST_D)
      AND h.tran_seq_no = e.tran_seq_no(+)
      AND h.store = e.store (+)
      AND h.day = e.day (+)
      AND e.status = :SAST_P
      AND e.system_code = :SYSE_RDW
ORDER BY 3;
```

The third driving cursor is encapsulated in the **getBalTotal** function, which fetches all totals with a usage_type of 'RDW'. It returns, among other things, the total_id, the cashier id and the register id. These are then used to determine whether to write a record to the RDWS file or the RDWC file. Only totals with a total_id of "OVRSHT_B" (over/short balance level) are exported to the RDWC file. The other totals are exported to the RDWS file only if both their register and their cashier ids are empty, i.e. the total is at the store level. If the total cannot be written to neither the RDWC nor the RDWS file, then we write an error to the log and continue.

## Performance

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

## Security Considerations

Credit card numbers and other customer information are present in the output files. Access to these files is controlled only by the Unix permissions that these files have.

# Sales Audit Export to RMS [saexprms]

## Purpose

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

## Design Overview

The purpose of this batch module is to fetch all corrected sale and return transactions that do not have RMS errors from the Retek Sales Audit (ReSA) database tables for transmission to the Retek Merchandising System (RMS). If sa_system_options.unit_of_work is 'S', then the whole store/day is skipped if any RMS error is found. If this value is 'T', then only transactions with RMS errors are skipped. If the transaction has a status of Deleted and it has previously been transmitted, a reversal of the transaction will be sent.

If the transaction has a status of 'D'eleted and it has previously been transmitted, a reversal of the transaction will be sent.

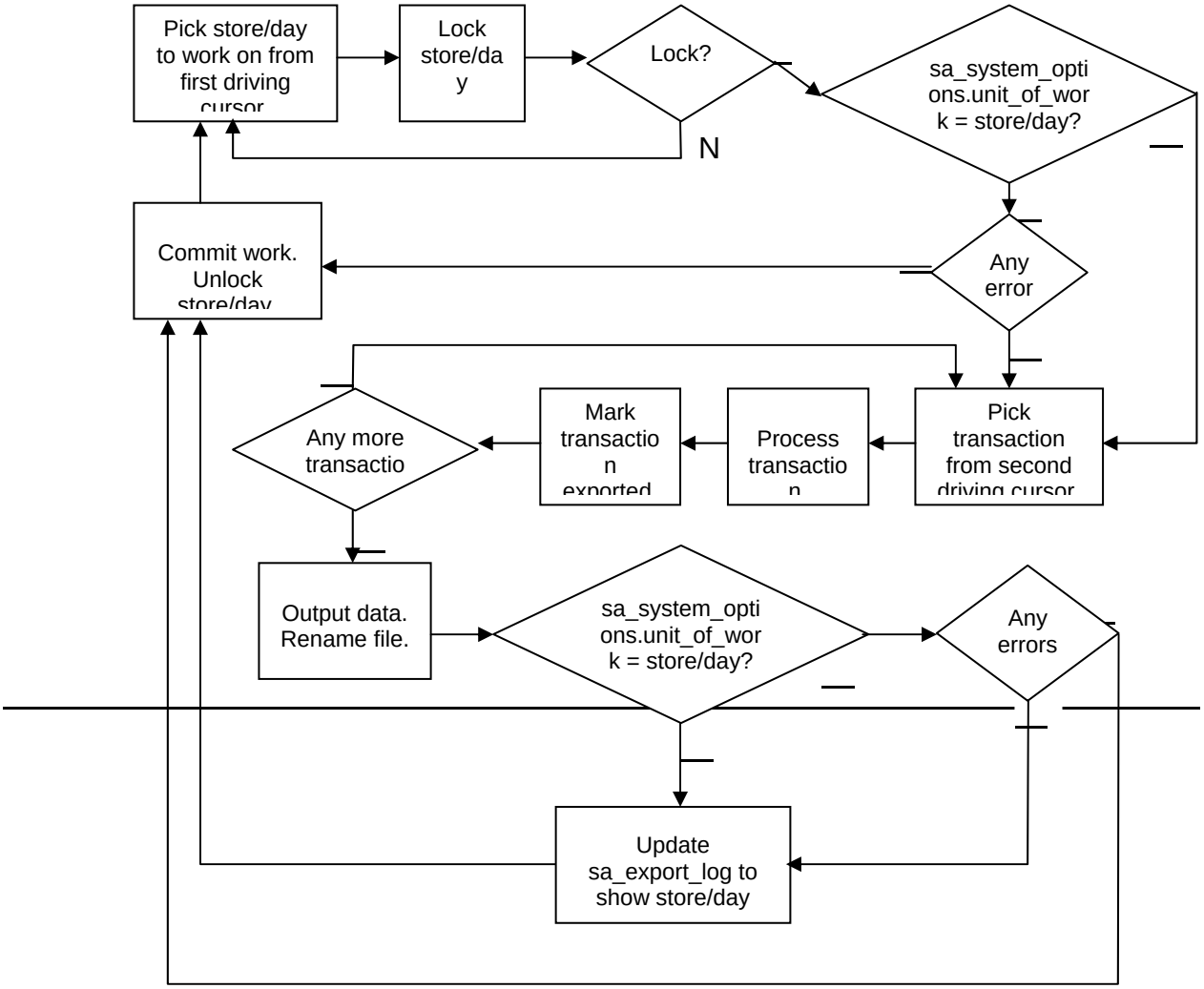
Multi-threading based on store was added to this program in version 11.0.6.

A file of type POSU is generated for each store/day.

**Tables Affected:**

| TABLE | SELECT | INSERT | UPDATE | DELETE |
|---|---|---|---|---|
| sa_system_options | Yes | No | No | No |
| sa_store_day | Yes | No | No | No |
| sa_export_log | Yes | No | No | No |
| sa_error | Yes | No | No | No |
| sa_tran_head | Yes | No | No | No |
| sa_tran_item | Yes | No | No | No |
| sa_tran_disc | Yes | No | No | No |
| sa_tran_head_rev | Yes | No | No | No |
| sa_tran_item_rev | Yes | No | No | No |
| sa_tran_disc_rev | Yes | No | No | No |
| store | Yes | No | No | No |
| currencies | Yes | No | No | No |
| sa_exported | Yes | No | No | No |
| sa_exported_rev | Yes | No | No | No |

## Program Flow




**Global Variable Descriptions**

| Global Variable | Description |
|---|---|
| Pi_commit_max_ctr | Commit max counter used for array fetch |
| ps_num_threads | Commit max counter used for array fetch |
| ps_thread_val | Commit max counter used for array fetch |
| pl_multiplier | Multiplier to remove decimals from numbers |
| Ps_unit_of_work | Unit of Work from sa_system_options. |
| Ps_sysdate | Current sysdate value from the database. |
| Ps_temp_file | Temporary file name to be used for the RMS file. |
| Ps_file_name | Final file name of the RMS output file. |
| *RMSoutFile | RMS output file |
| FileRecOutCount | Record count out for FTAIL. |

| Global Variable | Description |
|---|---|
| pi_tdetl_count | Tdetl record count for ttail |
| Pi_tdetl_count | TDETL record count for TTAIL record in the RMS file. |
| Current_item | Pointer to the current item node of the binary tree. |
| Itemroot | Root of the binary tree. |
| RMSoutFile | RMS output file pointer. |
| Ps_store | Current store/location ID. |
| Ps_business_date | Business date of store/day combination |
| Ps_vat_region | VAT Region for the current store/day. |
| Ps_vat_include_ind | VAT Include Indicator for the current store/day. |
| Ps_currency_code | Currency Code for the current store/day. |
| Ps_currency_rtl_dec | Currency Retail Decimal places for the current store/day. |
| Pi_num_locks_not_released | Counter for the number of store/day locks that could not be released. |
| pa_sa_tran_item | Array for tran_item record |
| pl_num_sa_tran_item | Size of tran_item record |
| pa_sa_tran_disc | Array for tran_disc record |
| pl_num_sa_tran_disc | Size of tran_disc record |

## Function Level Description

**main()**

int argc

char *argv[]

Check command line for required arguments.

Call **LOGON** to connect to the database.

Call **Init** to initialize the program.

Call **process** to export the available RMS data.

Report unlocking errors.

Call **final** to cleanup.

**init()**

No arguments

This function initializes Restart recovery.

Initialize OraNum functions by calling OraNumInit.

Get the value of sa_system_options.unit_of_work by calling the library function fetchSaSystemOptions.

Get a temporary filename to use for generating the output files. Store this name in ps_temp_file.

#### process()

No arguments

Picks a store/day to be processed by using the first driving cursor.

Multi threading is implemented using Store as the threading scheme.

Attempt to lock the store/day with a call to get_lock. If this fails, go on to the next store/day.

If sa_system_options.unit_of_work is store/day, than check to see if any of the store/days transactions have RMS errors by calling the library function fetchStoreDayErrorCount. If they do, unlock the store/day with a call to release_lock and go on to the next store/day.

Get VAT information by calling fetchVatCur.

Call fetchSysDate to get the current date/time.

Save the store/location ID in ps_store.

Initialize itemroot and current_item to NULL.

Call processStoreDay to process the store/days transactions.

Call WrOutputData to format and write the RMS output records.

Update the status in sa_export_log to 'E'xported 'C'omplete by calling the library function markStoreDayExported

Rename temporary output file to posu_*store_businessdate_curdatetime*.

Call to release_lock and go on to the next store/day. This function commits as a side effect, thus committing the changes to the database.

#### final()

No arguments

Remove the temporary file.

Call retek_close.

#### FetchVatCur()

Char is_store_day_seq_no[NULL_BIG_SEQ_NO]

Call a cursor that will retrieve the vat_include_ind, vat_region, currency_code and currency_rtl_dec. Gets vat_include_ind, vat_region, currency_code, currency_rtl_dec from sa_store_day and store tables for RMSFHead record in output file.

#### processStoreDay()

char *is_store_day_seq_no

char *is_store

char *is_day

char *is_sysdate

For each transaction from the store/day being processed, get the following information from the second driving cursor and call **processTransHead** with the information.

| Table | Column | Description |
|---|---|---|
| Sa_tran_head | Tran_seq_no | |
| Sa_tran_head | Rev_no | |
| Sa_tran_head | Tran_datetime | Format YYYYMMDDHH24MISS |

| Table | Column | Description |
|---|---|---|
| Sa_tran_head | Status | |
| | Transaction Sign | SAFD_P if the transaction has not been deleted (status != SAST_D) and there are no errors and it has not been exported.<br>SAFD_N if the transaction has been deleted (status = SAST_D) and it has been exported after being exported. |
| Sa_tran_head | Sub_tran_type | |
| Sa_tran_head | Tran_type | |

Calls the library function **markTransactionExported** to insert a record into sa_exported.

#### processTransHead()

struct pt_sa_tran_head *ir_sa_tran_head

char is_store_day_seq_no[NULL_BIG_SEQ_NO]

char *is_store

char *is_day

If the transaction status is deleted (SAST_D) and it has been previously exported, then call **retrieveTransHeadRev.** Also, if the revision number of the transaction is not 1, then a previous revision may have been exported; call **retrieveTransHeadRev** to get the exported revision (for full disclosure purposes).

Call **retrieveTransItem** and **retrieveTransDisc** to obtain the items and discounts for the transaction, both Positive transactions and Negative ones.

Call **saveData** for both the Positive and Negative transactions to write the information into memory.

#### retrieveTransHeadRev()

char *is_tran_seq_no

char *is_store

char *is_day

struct pt_sa_tran_head *or_sa_tran_head_rev

This function gets the sa_tran_head_rev record that needs to be processed. A record needs to be processed if it has been previously exported.

| Table | Column | Description |
|---|---|---|
| Sa_tran_head_rev | Tran_seq_no | |
| Sa_tran_head_rev | Rev_no | |
| Sa_tran_head_rev | Tran_datetime | Format YYYYMMDDHH24MISS |
| Sa_tran_head_rev | Status | |
| | Transaction Sign | SAFD_N |
| Sa_exported_rev | Exp_datetime | Only for transactions with a Transaction Sign of SAFD_N.<br>Format YYYYMMDDHH24MISS |
| Sa_exported_rev | Sub_tran_type | |

If no data is found, then set or_sa_tran_head_rev->s_rev_no to –1.

**retrieveTransItem()**

char *is_tran_seq_no,
long *ol_num_sa_tran_item,
struct pt_sa_tran_item **oa_sa_tran_item

This function gets all the tran_item records for a tran_seq_no that needs to be processed.

**retrieveTransItemRev()**

char *is_tran_seq_no,
char *is_store,
char *is_day,
char *is_rev_no,
long *ol_num_sa_tran_item,
struct pt_sa_tran_item **oa_sa_tran_item

This function gets all sa_tran_item records or sa_tran_item_rev (if is_rev_no is not –1) that need to be processed for a tran_seq_no.

Records should be limited to those with an item_type of SAIT_ITEM and SAIT_REF.

| Table | Column | Description |
|---|---|---|
| Sa_tran_item_rev | Tran_seq_no | |
| Sa_tran_item_rev | Item_seq_no | |
| Sa_tran_item_rev | Item_status | |
| Sa_tran_item_rev | Item_type | |
| Sa_tran_item_rev | Item | |
| Sa_tran_item_rev | Ref_item | |
| Sa_tran_item_rev | Dept | |
| Sa_tran_item_rev | Class | |
| Sa_tran_item_rev | Subclass | |
| Sa_tran_item_rev | Pack_ind | |
| Sa_tran_item_rev | Item_level | |
| Sa_tran_item_rev | Tran_level | |
| Sa_tran_item_rev | Waste_type | |
| Sa_tran_item_rev | Waste_pct | Value multiplied by 10000. |
| Sa_tran_item_rev | Qty | Absolute value multiplied by 10000. |
| Sa_tran_item_rev | Unit_retail | Absolute value multiplied by 10000. |
| Sa_tran_item_rev | Selling_uom | |
| Sa_tran_item_rev | Drop_ship_ind | |
| Sa_tran_item_rev | Catchweight_ind | |
| Item_master | Uom_quantity | |

The same columns as above are selected from the sa_tran_item_rev table if the rev_no passed in is not –1.

Set *ol_num_sa_tran_item to the total number of records fetched.

**retrieveTransDisc()**

char *is_tran_seq_no

long *ol_num_sa_tran_disc

struct pt_sa_tran_disc **oa_sa_tran_disc

This function gets the tran_disc records for a tran_seq_no that needs to be processed.

**retrieveTransDiscRev()**

char *is_tran_seq_no,

char *is_store,

char *is_day,

char *is_rev_no,

long *ol_num_sa_tran_disc,

struct pt_sa_tran_disc **oa_sa_tran_disc

This function gets all sa_tran_disc or sa_tran_disc_rev records (if is_rev_no is not –1) for a tran_seq_no that needs to be processed.

Records should be limited to those with an item_type of SAIT_ITEM and SAIT_REF.

| Table | Column | Description |
|---|---|---|
| Sa_tran_disc_rev | Tran_seq_no | |
| Sa_tran_disc_rev | Item_seq_no | |
| Sa_tran_disc_rev | rms_promo_type | |
| Sa_tran_disc_rev | Promotion | |
| Sa_tran_disc_rev | Discount_type | |
| Sa_tran_disc_rev | Qty | SIGN_N or SIGN_P depending on the sign of qty. |
| Sa_tran_disc_rev | Qty | Absolute value multiplied by 10000. |
| Sa_tran_disc_rev | Unit_discount_amt | Value multiplied by 10000. |
| Sa_tran_disc_rev | Promo_comp | |

The same columns as above are selected from the sa_tran_disc_rev table if the rev_no passed in is not –1.

Set *ol_num_sa_tran_disc to the total number of records fetched.

**saveData()**

struct pt_sa_tran_head *ir_sa_tran_head

long il_num_sa_tran_item

struct pt_sa_tran_item *ia_sa_tran_item

long il_num_sa_tran_disc

struct pt_sa_tran_disc *ia_sa_tran_disc

Set ll_cur_sa_tran_item and ll_cur_sa_tran_disc to 0.

For each item record:

Call **Blank_field** to blank pad NULL values so there would be no zeros in the VRTLOG.

Call **AddItem** to add the new item to the tree.

For each item's discount record:
Call **AddDiscData** to add discount data onto the existing item.

**populateArrays**

char *os_store_day_seq_no,
long *ol_num_sa_tran_item,
struct pt_sa_tran_item **oa_sa_tran_item,
long *ol_num_sa_tran_disc,
struct pt_sa_tran_disc **oa_sa_tran_disc

This function will fetch all required information for a store_day from the sa_tran_item and sa_tran_disc tables and populate the global arrays with the data.

| Table | Column | Description |
|---|---|---|
| Sa_tran_item | Tran_seq_no | |
| Sa_tran_item | Item_seq_no | |
| Sa_tran_item | Item_status | |
| Sa_tran_item | Item_type | |
| Sa_tran_item | Item | |
| Sa_tran_item | Ref_item | |
| Sa_tran_item | Dept | |
| Sa_tran_item | Class | |
| Sa_tran_item | Subclass | |
| Item_master | Pack_ind | |
| Item_master | Item_level | |
| Item_master | Tran_level | |
| Sa_tran_item | Waste_type | |
| Sa_tran_item | Waste_pct | Value multiplied by 10000. |
| Sa_tran_item | Qty | Absolute value multiplied by 10000. |
| Sa_tran_item | Unit_retail | Absolute value multiplied by 10000. |
| Sa_tran_item | Selling_uom | |
| Sa_tran_item | Drop_ship_ind | |
| Sa_tran_item | Catchweight_ind | |
| | Uom_quantity | Value multiplied by 10000 |

| Table | Column | Description |
|---|---|---|
| Sa_tran_disc | Tran_seq_no | |
| Sa_tran_disc | Item_seq_no | |
| Sa_tran_disc | rms_promo_type | |
| Sa_tran_disc | Promotion | |

| Table | Column | Description |
|---|---|---|
| Sa_tran_disc | Discount_type | |
| Sa_tran_disc | Qty | Value multiplied by 10000. |
| Sa_tran_disc | Unit_discount_amt | Value multiplied by 10000. |
| Sa_tran_disc | Promo_comp | |

For each item record:

Set current_item to NULL.

Call c_sa_tran_item to retrieve transaction records from sa_tran_item.

If no data is found set li_end to 1.

Reset ll_alloc_size, ll_records_to_process and li_end to 0.

For each item record:

Call c_sa_tran_disc to retrieve transaction records from sa_tran_disc.

If no data is found set li_end to 1.

Processes a RTLFTail record from the input file.

Calls delete_item_tree to delete all the items in memory.

#### AddItem()

RTL_TITEM *tip

Struct pt_sa_tran_head *ir_sa_tran_head

Struct pt_sa_tran_item *irs_sa_tran_item

Finds an item or adds a new item to the tree. Returns pointer if OK, or NULL if failure.

If item_status is SASI_V, then it needs to be reset depending on the sign of qty. Either SASI_R if it is negative or SASI_S if it is positive.

**Nextitem** is called to do the rollups that happen when a subsequent item of the same type is encountered.

If the item is not found, than **newitem** is called to create it and add it to the tree.

#### Newitem()

char *pricepoint

int hv

Struct pt_sa_tran_head *ir_sa_tran_head

Struct pt_sa_tran_item *irs_sa_tran_item

This function allocates memory to a new item and returns a pointer or NULL.

#### Nextitem()

Struct pt_sa_tran_head *ir_sa_tran_head

Struct pt_sa_tran_item *irs_sa_tran_item

Struct ITEM_TAG *item_tag_ptr

This function performs the rollups when a subsequent item of the same type is encountered.

#### AddDiscData()

Struct pt_sa_tran_head *ir_sa_tran_head

Struct pt_sa_tran_disc *ir_sa_tran_disc

char is_item_status[NULL_CODE]

Struct ITEM_TAG *i_item

Adds discount data into the existing item.

Calls finddiscdata to find a discount or adds a new discount to the item. Returns pointer to the data if OK, or NULL if failure. Nextdiscdat is called to do the rollups that happen when a subsequent discount of the same type is encountered.

If the discount is not found, than **newdiscdat** is called to create it and add it to the item.

#### Finddiscdata()

struct DISCDAT_TAG *discdat_tag_ptr

Char *is_type

Char *is_amt

Char *is_promotion

Searches the ITEM for the matching discount type.

#### Nextdiscdat()

Struct ITEM_TAG *item_tag_ptr

struct DISCDAT_TAG *discdat_tag_ptr

char is_item_status[NULL_CODE]

char is_tran_sign[NULL_IND]

Struct pt_sa_tran_disc *ir_sa_tran_disc

This functionnextitem does the rollups that happen when an subsequent item of the same type is encountered. These 2 functions nee d to update the last_time_modified field each time they are called. The value for this field will come from the FHEAD and the THEAD records. The FHEAD record contains the date portion and the THEAD record contains the time portion.

performs the rollups when a subsequent discount of the same type is encountered.

Does the rollups that happen when a subsequent discount of the same type is encountered.

#### Newdiscdat()

Struct ITEM_TAG *item_tag_ptr

char is_item_status[NULL_CODE]

char is_tran_sign[NULL_IND]

Struct pt_sa_tran_disc *ir_sa_tran_disc

This function allocates memory to a discdat node and returns a pointer or NULL.

#### WrOutputData()

No arguments

Open the RMS output temporary file (ps_temp_file).

Write a RMS FHEAD record by calling WrRMSFHead.

Write the RMS transaction records by calling wod.

Write a RMS FTAIL record by calling WrRMSFTail.

Close the RMS temporary output file.

#### Wod

Struct ITEM_TAG *item_tag_ptr

Calls itself recursively to output data from the entire binary tree.

If item_tag_ptr is not NULL

Call wod to recurse down the left branch.

If the total negative sales quantity is not zero than process the reverse sale.

Call WrRMSTHead to write the THEAD record.

For each of the discount records attached to this item, call WrRMSTDetl to write TDETL.

Call WrRMSTTail to write the reversed sales TTAIL record.

If the total positive sales quantity is not zero than process the positive sale.

Call WrRMSTHead to write the THEAD record.

For each of the discount records attached to this item, call WrRMSTDetl to write TDETL.

Call WrRMSTTail to write the reversed sales TTAIL record.

If the total negative return quantity is not zero than process the reverse return.

Call WrRMSTHead to write the THEAD record.

For each of the discount records attached to this item, call WrRMSTDetl to write TDETL.

Call WrRMSTTail to write the reversed sales TTAIL record.

If the total positive return quantity is not zero than process the positive return.

Call WrRMSTHead to write the THEAD record.

For each of the discount records attached to this item, call WrRMSTDetl to write TDETL.

Call WrRMSTTail to write the reversed sales TTAIL record.

Call wod to recurse down the right branch.

#### WrRMSFHead()

No arguments

Writes an RMS_FHEAD record (as defined in salib.h) to the specified output file. This must match the definition of the record in Interface File – SA to RRMS.doc.

Set FileRecOutCount to 1.

| Field | Type | Size | Source |
|---|---|---|---|
| Frecdesc | char | RMS_FRECDESC_SIZE | RMS_FHEAD_FRECDESC |
| Flineid | char | FT_NUMBER | ls_file_line_no |
| file_type_definition | char | FT_VARCHAR | 'POSU' |
| file_create_date | char | FT_DATE | ps_sysdate |
| Location | Char | FT_NUMBER | ps_store |
| Vat_include_ind | Char | FT_VARCHAR | ps_vat_include_ind |
| Vat_region | Char | FT_NUMBER | ps_vat_region |
| Currency_code | Char | FT_VARCHAR | ps_currency_code |
| Currency_rtl_dec | char | FT_NUMBER | ps_currency_rtl_dec |

Call **putrec** to write the record out to the RMS file.

#### WrRMSTHead()

Struct ITEM_TAG *item_tag_ptr

Const char is_tran_sign[NULL_CODE]

Char is_tran_sign[NULL_CODE]

This function writes a RMSTHead record to the output file. This function needs to copy the last_time_modified from the ITEM_TAG struct into the RMS_THEAD struct before calling **putrec**.

| Field | Type | Size | Source |
|---|---|---|---|
| Frecdesc | char | RMS_FRECDESC_SIZE | RMS_THEAD_FRECDESC |
| Flineid | char | FT_NUMBER | ls_file_line_no |
| Business_date | char | FT_DATE | ps_business_date |
| Item_type | char | FT_VARCHAR | item_tag_ptr->item_type |
| Item | char | FT_VARCHAR | item_tag_ptr->item |
| Dept | char | FT_NUMBER | item_tag_ptr->dept |
| Class | char | FT_NUMBER | item_tag_ptr->class |
| Subclass | char | FT_NUMBER | item_tag_ptr->subclass |
| Pack_ind | char | FT_VARCHAR | item_tag_ptr->pack_ind |
| Item_level | char | FT_NUMBER | item_tag_ptr->item_level |
| Tran_level | char | FT_NUMBER | item_tag_ptr->tran_level |
| Waste_type | char | FT_NUMBER | item_tag_ptr->waste_type |
| Waste_pct | char | FT_NUMBER | item_tag_ptr->waste_pct |
| Selling_uom | char | FT_VARCHAR | item_tag_ptr->selling_uom |
| Drop_ship_ind | char | FT_VARCHAR | item_tag_ptr->drop_ship_ind |
| Tran_type | char | FT_VARCHAR | SASI_S if is_item_status = SASI_S else SASI_R |
| Catchweight_ind | char | FT_VARCHAR | item_tag_ptr->catchweight_ind |
| Substrans_type_ind | char | FT_VARCHAR | item_tag_ptr->subtrans_type_ind |

#### WrRMSTDetl()

Struct DISCDAT_TAG *discdat_tag_ptr

Writes a RMSTDetl record to the output file.

#### WrRMSTTail()

No arguments

Writes a RMSTTail record to the output file.

#### WrRMSFTail()

No arguments

Writes a RMSFTail record to the output file.

#### Alloc_item()

No arguments

This allocates memory for an item node.

**delete_item_tree()**

struct ITEM_TAG *itemroot

This function recursively deletes all items in memory.

Calls delete_discounts to delete discount information from the tree (before deleting the current node).

**Alloc_discount()**

No arguments

This allocates memory on discounts for a particular item.

**delete_discounts ()**

struct DISCDAT_TAG *discdat_tag_ptr

This function recursively deletes all discounts for a particular item in memory.

**Blank_field ()**

char *is_field

int ii_len

This function fills the character array with spaces up to ii_len.

**log_and_exit ()**

char *is_message

This function writes is_message to the message log, calls final() and then exits.

## Stored Procedures / Shared Modules (Maintainability)

| Shared Module | Module Description |
|---|---|
| libretek.a functions | Refer to Library Design – retek.doc for details. |
| retek_init | Initialize restart recovery. |
| retek_close | Close restart recovery functions. |
| Retek_refresh_thread | Refresh the current thread so that it may be used again. |
| | |
| Libresa.a functions: | Refer to Library Design – ReSA.doc for details. |
| get_lock | Used to establish a read lock on a store/day. |
| release_lock | Used to release a store/day lock. |
| fetchSaSystemOptions | Fetch the values from the sa_system_options table. |
| fetchSysDate | Fetch the current SYSDATE value. |
| fetchStoreDayErrorCount | Fetch the number of errors that corresponds to a particular store/day and system. |
| markStoreDayExported | Mark a particular store/day and system as exported |
| markTransactionExported | Mark a particular transaction and system as exported. |
| OraNum functions (Add, Sub, Mul, Div) | Used to perform arithmetic operations on strings containing large numbers. |

| Shared Module | Module Description |
|---|---|
| Putrec | Writes a record to a file. |

## Input Specifications

There are 2 driving cursors in this module. The first picks a store/day to work on:

```
   SELECT sd.store_day_seq_no,
          el.seq_no,
          sd.store,
          sd.day,
          TO_CHAR(sd.business_date, 'YYYYMMDD'),
          sd.data_status,
          ROWIDTOCHAR(el.rowid)
     FROM sa_store_day sd, sa_export_log el, v_restart_store vrs
    WHERE sd.store_day_seq_no = el.store_day_seq_no
      AND sd.store = el.store
      AND sd.day = el.day
      AND sd.store_status IN (:SASS_W, :SASS_C) /* Worksheet or Closed       */
      AND sd.data_status IN (:SADS_P, :SADS_F)  /* Partially or Fully loaded */
      AND el.system_code = :SYSE_RMS
      AND el.status = :SAES_R                   /* 'R'eady to be exported    */
      AND vrs.num_threads = TO_NUMBER(:ps_num_threads)
      AND vrs.thread_val = TO_NUMBER(:ps_thread_val)
      AND vrs.driver_value = sd.store
 ORDER BY sd.store_day_seq_no, sd.store, sd.business_date;
```

The second fetches the transaction data to be output:

```
SELECT h.tran_seq_no,
            LTRIM(h.rev_no, '0'),
            TO_CHAR( h.tran_datetime, 'YYYYMMDDHH24MISS'),
            h.status,
            :SAFD_P,
            ' ',
            nvl(h.sub_tran_type,' '),
            h.tran_type
       FROM sa_tran_head h
      WHERE h.store_day_seq_no = TO_NUMBER(:is_store_day_seq_no)
        AND h.store = TO_NUMBER(:is_store)
        AND h.day = TO_NUMBER(:is_day)
        AND ( (h.tran_type IN (:TRAT_SALE, :TRAT_RETURN, :TRAT_EEXCH)
              OR (h.tran_type = :TRAT_PAIDOU and h.sub_tran_type =
:TRAS_CACCOM)))
         AND (h.status = :SAST_P
              AND NOT EXISTS            /* and no errors for the transaction */
                 (SELECT er.tran_seq_no
                    FROM sa_error er, sa_error_impact ei
                   WHERE h.tran_seq_no = er.tran_seq_no
                     AND h.store = er.store
                     AND h.day = er.day
                     AND er.error_code = ei.error_code
                     AND ei.system_code = :SYSE_RMS
                     AND er.hq_override_ind != :YSNO_Y))
        AND NOT EXISTS
            (SELECT e.store_day_seq_no
               FROM sa_exported e
              WHERE h.store_day_seq_no = e.store_day_seq_no
                AND h.store = e.store
                AND h.day = e.day
                AND h.tran_seq_no = e.tran_seq_no
```

```
                AND e.system_code = :SYSE_RMS)
   UNION ALL
      SELECT h.tran_seq_no,
             LTRIM(h.rev_no, '0'),
             TO_CHAR( h.tran_datetime, 'YYYYMMDDHH24MISS'),
             h.status,
             :SAFD_N,
             NVL( TO_CHAR( e.exp_datetime, 'YYYYMMDDHH24MISS'), ' '),
             nvl(h.sub_tran_type,' '),
             h.tran_type
        FROM sa_tran_head h,
             sa_exported e
       WHERE h.store_day_seq_no = TO_NUMBER(:is_store_day_seq_no)
         AND h.store = TO_NUMBER(:is_store)
         AND h.day = TO_NUMBER(:is_day)
         AND ( (h.tran_type IN (:TRAT_SALE, :TRAT_RETURN, :TRAT_EEXCH)
               OR (h.tran_type = :TRAT_PAIDOU and h.sub_tran_type =
:TRAS_CACCOM)))
         AND h.status IN (:SAST_D, :SAST_V)
         AND h.tran_seq_no = e.tran_seq_no(+)
         AND h.store = e.store
         AND h.day = e.day
         AND e.status = :SAST_P
         AND e.system_code = :SYSE_RMS
       ORDER BY 3;
```

## Output Specifications

Data is output in the POSU file format. This is described in Interface File – SA to RMS.doc.

The filename convention for these valid POSU files will be posu_*store_businessdate_curdatetime*. The file should start out with a temporary name generated by the Unix tempnam (See Unix man page 3S) call and then be renamed with Unix rename (See Unix man page 2) call when the file is complete.

## Database Integrity

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

## Parameter Validation

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

## Integrity Constraints

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

## Scheduling Considerations

Processing Cycle: Anytime – Sales Audit 10.0 is a 24/7 system.

Scheduling Diagram: This program will be run after auditors have made corrections to the data. This module should not be run simultaneously with other modules: saexprdw, saexpim, saexpuar, saexpach, and saexpgl.

Pre-Processing: sagetref.pc to get reference data.

Post-Processing: posupld.pc should be run after saexprms.pc to import the data into the RMS system.

Threading Scheme: v_restart_store

## Locking Strategy

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

### Restart / Recovery

The logical unit of work for this module is defined as a unique store/day combination. Records will be fetched, updated and inserted in batches of pl_commit_max_ctr. Only two commits will be done, one to establish the store/day lock and another at the end, to release the lock after a store/day has been completely processed. The POSU formatted output file will be created with a temporary name and renamed just before the end of store/day commit.

In case of failure, we rollback all work done to the point right after the call to get_lock() and then we release the lock. Thus, we assume that the rollback segment is large enough to hold all inserts into sa_exported for one store_day. If this is not the case, we need to increase the size of the rollback segment. The EXEC SQL SAVEPOINT statement is used to save the state of the database after the call to get_lock().

## Sales Audit Export to UAR [saexpuar]

### Functional Area

Universal Account Reconciliation - UAR Export

### 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

### 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. This module should not be run simultaneously with other modules: saexprms, saexprdw, saexpim, saexpach, and saexpgl.

Threading Scheme: N/A

## Restart Recovery

Logical Unit of Work ( recommended Commit checkpoints )

Driving Cursor

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

#### Driving cursor 1:

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

#### Driving Cursor 2:

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

## Program Flow

Structure Chart

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

Functional Design – SA export.doc

## Shared Modules

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

libresa library functions:

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

## Function Level Description

All database interactions required and error handling considerations

#### init ()

1. Call **OraNumInit** to initialize string numbers arithmetic operations.
2. Get the current system date from the library function **fetchSysdate**.
3. Get the unit-of-work by calling the library function **fetchSaSystemOptions**.

#### process ()

1. Loop through the libresa library function **fetchStoreDayToBeExportedLike**.
2. Attempt to obtain a read lock on the store/day with a call to **get_lock**. If this fails, go on to the next store/day and log the problem to the error log.
3. Call the function processStoreDay for the current store day.
4. Call the function **markStoreDayExported**.
5. Call the function **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

### All files layouts input and output

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

- writeStoreDayHeader
- writeStoreDayDetail
- writeStoreDayFooter

# Stock Ledger Append [salapnd]

## Design Overview

The sole purpose of this program is to move data from the transaction staging table into the historical transaction table. This requires placing a lock on the staging table to ensure that no new data will be added to it while the movement is occurring (to handle trickling or real-time processing), moving the data to the historical table.

## Scheduling Constraints

Processing Cycle: Should occur after all extractions have completed (RMS – saldly, RDW etc.)

Scheduling Diagram: N/A

Pre-Processing: salstage, all extraction, and all processing

Post-Processing: N/A

Threading Scheme: Threading will be implicit via the use of the Oracle Parallel Query Option. The insert/select query should be tuned for each specific environment to achieve the best throughput.

## Restart Recovery

No specific restart/recovery scheme needs to be defined because of the limited scope of the module. However, in cases where the tran_data table is very large, a particularly large rollback segment may be specified to reduce the risk of exceeding rollback segment space. This will depend on the size of normal rollback segments and the size of the tran_data table.

## Program Flow

N/A

## Shared Modules

N/A

## Function Level Description

**Init():**

The rollback segment should be specified to a large enough size so that the entire rollup will complete in a single transaction.

**Process()**

This function first calls the lock_table function to lock the if_tran_data table. It then calls the append_history function to append the data from if_tran_data on to the tran_data_history table. Finally, it calls lock table to release the lock on the tran_data table.

**Lock_table()**

This function places or remove a lock on a table passed in. It will place or release the lock depending on a input parameter.

**Append_history()**

This function appends the data on if_tran_data on to the tran_data_history table.. This is done using an insert/select statement with a hint for parallelism. The degree of parallelism should be customized to each run-time environment.

## I/O Specification

N/A