Oracle® Retail Merchandising System

Operations Guide Addendum

Release 10.1.21

February 2008

Oracle® Retail Merchandising System Operations Guide Addendum, Release 10.1.21

Copyright © 2008, Oracle. All rights reserved.

Primary Author: Nathan Young

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software—Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

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 **WebLogic™** developed and licensed by BEA Systems, Inc. of San Jose, California, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

(x) 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
- Functional integration dataflow across the enterprise

This Operations Guide Addendum should be used in conjunction with previously released Oracle Retail Merchandising System 10.x documentation.

Audience

Anyone with an interest in developing a deeper understanding of the underlying processes and architecture supporting Oracle Retail Merchandising System 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 and other systems across the enterprise.
- System analysts and system operations personnel:
  - Who are looking for information about Oracle Retail Merchandising System's processes internally or in relation to the systems across the enterprise.
  - Who operate Oracle Retail Merchandising System regularly.
- Integrators and implementation staff with overall responsibility for implementing Oracle Retail Merchandising System.

Related Documents

For more information, see the following documents in the Oracle Retail Merchandising System Release 10.1.21 documentation set:

- Oracle Retail Merchandising System Release Notes
- Oracle Retail Merchandising System Installation Guide
- 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

The information in this document reflects modifications and updates to the *Oracle Retail Merchandising System 10.0 Operations Guide* and any subsequent RMS 10.x.x Operations Guide Addendums. Using this document in conjunction with the *Oracle Retail Merchandising System 10.0 Operations Guide* provides retailers with a complete overview of the application.

For the RMS 10.1.21 release, there is one new batch design (saimptlogtdup_upd) and two updated batch designs (prchstprg.pc and saimptlog). For more specific information regarding enhancements and modifications made to the previous Oracle Retail Merchandising System release, see the *Oracle Retail Merchandising System 10.1.21 Release Notes*.

2 Batch Designs

Retailers should refer to these sections in lieu of the corresponding batch designs in the RMS 10.0 Operations Guide or any subsequent RMS 10.x.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.

prchstprg.pc (Purge Price History)

Functional Area

Pricing

Module Affected

PRCHSTPRG.PC

Design Overview

The PRCHSTPRG program deletes price_hist records, which are older than a number of retention days specified in a new column added to system_options table as system_options.price_hist_retention_days. This program keeps the latest record for the combination of item, location and tran type and deletes the rest of the records, which fall in the specified period of retention days.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	PHASE AD-HOC (daily)
Scheduling Considerations	This program is run prior to phase 3 to improve select operations.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	Multi threaded. Threaded by table partition

Restart/Recovery

This program will use the commit_max_ctr on the restart_control table to periodically commit SQL delete operations. Restart/Recovery is achieved by processing records that have not been deleted. Table restart_bookmark stores the ps_cur_restart_partition_position for partition position as bookmark_string to restart a thread.

However, in cases where the price_hist 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 price_hist table.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

The commit_max_ctr field should be set to prevent excessive rollback space usage, and to reduce the overhead of file I/O. The recommended commit counter setting is 10000 records (subject to change based on experimentation). In case price_hist table is very large then the number of partitions on the table may be increased and then after the number of threads for this program should be increased.

Key Tables Affected

Table	Select	Insert	Update	Delete
PERIOD	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No
PRICE_HIST	No	No	No	Yes
DBA_TAB_PARTITIONS	Yes	No	No	No

Program Flow

N/A

Program Level Description

init():

Fetches system variable system_options.price_hist_retention_days from system_options table and vdate from the period table.

process():

Fetches partition_name and partition_position from DBA_TAB_PARTITIONS for table PRICE_HIST ,which is used in delete_history().

delete_history ():

Deletes records from price_hist in a specific partition, which are older than the date calculated backward from vdate for the value in price_hist_retention_days except the latest record for the combination of item, location and tran type, which fall in the specified period of retention days

I/O Specification

Output File Layout

N/A

saimptlog (Sales Audit Import)

Purpose

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

Objectives

This Batch Detailed Design must:

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

Functional Area

Sales Audit import

Module Affected

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

saimptlog.c
saimptlog.h
saimptlog_final.c
saimptlog_init.c
saimptlog_manval.c
saimptlog_nexttsn.pc
saimptlog_nextvhn.pc
saimptlog_output.c
saimptlog_uom.pc
saimptlog_proto.h
saimptlog_rtlog.c
saimptlog_sqlldr.c[¯]
saimptlog_tdup.c
saimptlog_loadtdup.c
saimptlog_tdup.h
saimptlog_nextmtsn.pc
saimptlog_nextesn.pc
saimptlog_ccval.c
saimptlog_ccval.h

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

- saimptlog_proto.h

SAIMPTLOGI

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

SAIMPTLOGFIN

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

Design Overview

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

SAIMPTLOG	SAIMPTLOGI
SAGETREF must be run to generate the current reference files: ▪ Items ▪ Wastage ▪ Sub-transaction level items ▪ Primary variant relationships ▪ Variable weight PLU ▪ Store business day ▪ Promotions ▪ Code types ▪ Error codes ▪ Credit card validation ▪ Store POS ▪ Tender type ▪ Merchant code types ▪ Partner vendors ▪ Supplier vendors ▪ Employee ids These files are all used as input to SAIMPTLOG and SAIMPTLOGI. Since SAIMPTLOG and SAIMPTLOGI can be threaded, this boosts performance by limiting interaction with the database.	
SAIMPTLOG is run against each POS file. SAIMPTLOG creates a write lock for store/day and than sets the data_status to loading until SAIMPTLOGFIN is executed. This generates distinct SQL*Loader files for that store/day for the sa_tran_head, sa_tran_item, sa_tran_disc, sa_tran_tax, sa_tran_tender, sa_error, sa_customer, sa_cust_attrib and (optionally) sa_missing_tran tables. A Retek formatted voucher file is produced for processing by SAVOUCH. SAIMPTLOG may be threaded as long as the parallel executions do not include the same store/day.	SAIMPTLOGI is run against each POS file. SAIMPTLOGI creates a write lock for that store/day and then sets the data_status to loading until SAIMPTLOGFIN is executed. This inserts data into sa_tran_head, sa_tran_item, sa_tran_disc, sa_tran_tax, sa_tran_tender, sa_error, sa_customer, sa_cust_attrib and (optionally) sa_missing_tran tables. A Retek formatted voucher file is produced for processing by SAVOUCH. SAIMPTLOGI may be threaded as long as the parallel executions do not include the same store/day.
SQL*Loader is executed to load the transaction tables from the files created by SAIMPTLOG. The store/day SQL*Loader files can be concatenated into a single file per table to optimize load times. Alternatively, multiple SQL*Loader files can be used as input to SQL*Loader. SQL*Loader may not be run in parallel with itself when loading a table. Header data (primary keys) must be loaded before ancillary data (foreign keys). This means that the sa_tran_head table must be loaded first; sa_tran_item before sa_tran_disc; and sa_customer before sa_cust_attrib. The remaining tables may be loaded in parallel.	
SAVOUCH is executed to load each of the Retek formatted voucher files. SAVOUCH may not be multiply threaded.	

SAIMPTLOG	SAIMPTLOGI
SAIMPTLOGFIN is executed to populate the sa_balance_group table, cancel post voided transactions and vouchers, validate missing transactions, and to mark the import as either partially or fully complete loaded. SAIMPTLOGFIN may not be multiply threaded.	

This design document encompasses SAIMPTLOG, SAIMPTLOGI and SAIMPTLOGFIN.

SAIMPTLOG

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

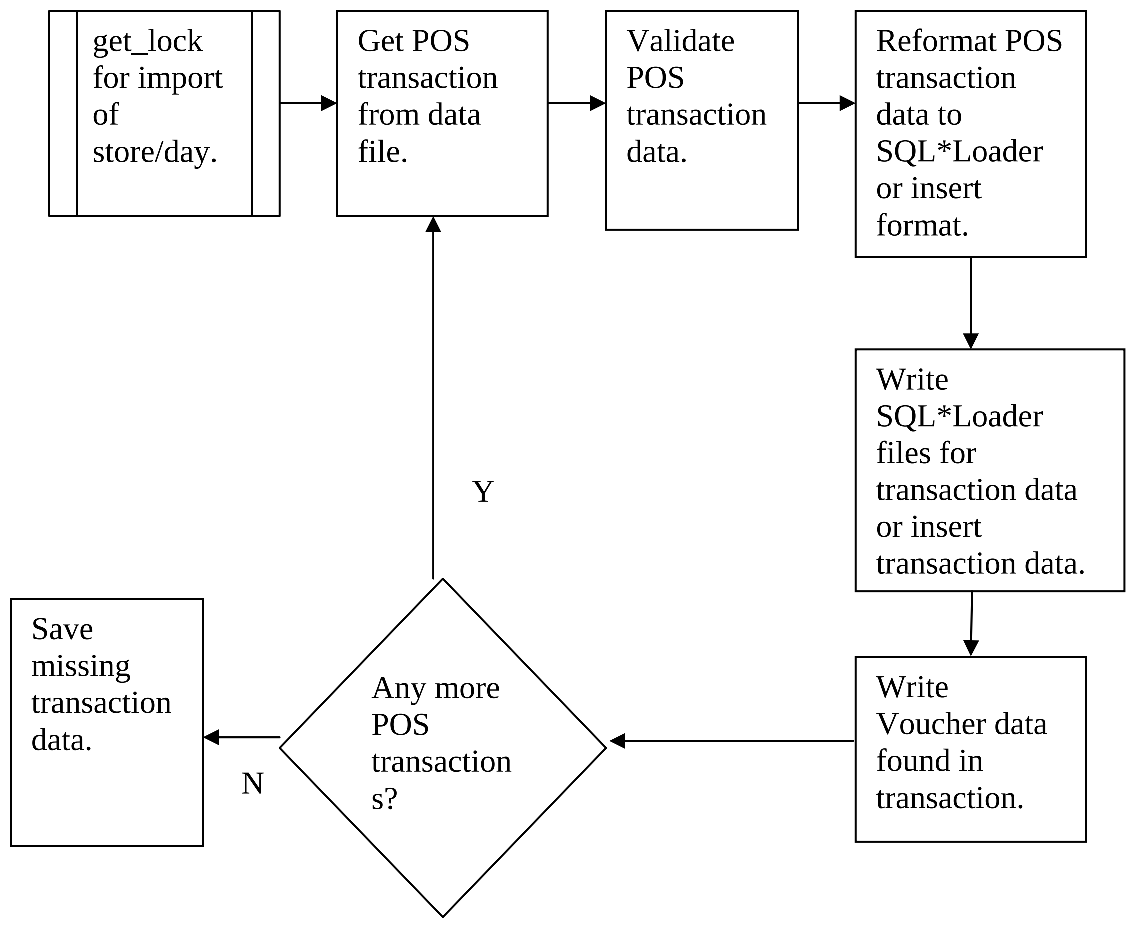
SAIMPTLOGI

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

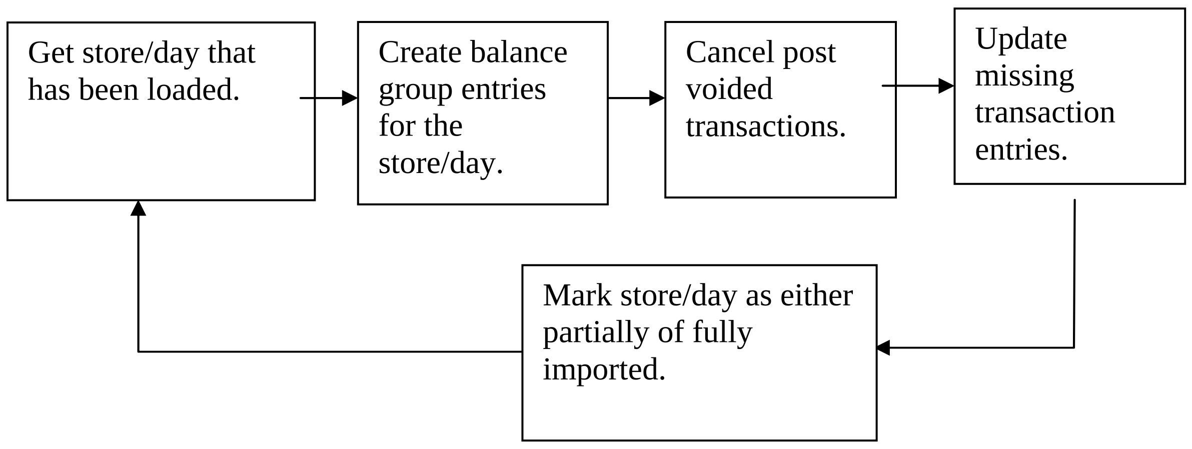
SAIMPTLOGFIN				
Table	Operations Performed			
	Select	Insert	Update	Delete
period	yes	no	No	no
store	yes	no	No	no
sa_system_options	yes	no	No	no
sa_store_day	yes	no	yes	no
sa_store_day_write_lock	yes	no	Yes	No
sa_import_log	yes	no	yes	no
sa_balance_group	Yes	Yes	No	No
sa_tran_head	Yes	No	Yes	Yes
sa_customer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

Program Flow

SAIMPTLOG and SAIMPTLOGI


get_lock for import of store/day.
Get POS transaction from data file.
Validate POS transaction data.
Reformat POS transaction data to SQL*Loader or insert format.
Write SQL*Loader files for transaction data or insert transaction data.
Write Voucher data found in transaction.
Any more POS transactions?
Y
N
Save missing transaction data.

SAIMTLOGFIN


Get store/day that has been loaded.
Create balance group entries for the store/day.
Cancel post voided transactions.
Update missing transaction entries.
Mark store/day as either partially of fully imported.

Function Level Description

SAIMPTLOG and SAIMPTLOGI

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

main() [saimptlog.c]

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

Process() [saimptlog.c]

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

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

Init() [saimptlog_init.c]

Call retek_init to initialize threading.

Get the system options by calling fetchSaSystemOptions.

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

Initialize the RTLOG file parser by calling InitInputData.

Load the item data generated by SAGETREF by calling item_loadfile.

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

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

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

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

Load the wastage data generated by SAGETREF by calling waste_loadfile.

Load the promotion data generated by SAGETREF by calling prom_loadfile.

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

Load the error data generated by SAGETREF by calling error_loadfile.

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

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

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

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

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

Load the employee data generated by SAGETREF by calling employee_loadfile.

Initialize transaction output processing by calling InitOutputData‡.

Initialize voucher output processing by calling openSAVoucher.

Initialize Oracle number arithmetic by calling OraNumInit.

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

Final() [saimptlog_final.c]

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

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

Terminate the RTLOG file parser by calling FinalInputData.

Call FinalOutputData‡ to finish any pending output processing.

Call closeSAVoucher to close and rename the voucher file.

Call retek_close to perform program status record keeping.

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

InitInputData() [saimptlog_rtlog.c]

Open the POS RTLOG file for reading.

Open a bad transaction file for writing.

Initialize the POS RTLOG transaction parser.

getNextTran() [saimptlog_rtlog.c]

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

Some record types will require some extra processing:

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

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

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

FinalInputData() [saimptlog_rtlog.c]

Close the POS RTLOG file.

Close the bad transaction file.

getRTLRec() [saimptlog_rtlog.c]

Read and return one record from the POS RTLOG file.

rtFind() [saimptlog_rtlog.c]

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

procRTLFHead() [saimptlog_rtlog.c]

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

procRTLFTail() [saimptlog_rtlog.c]

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

procRTLTHead() [saimptlog_rtlog.c]

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

Initialize the sale and tender transaction totals to 0.

procRTLTTail() [saimptlog_rtlog.c]

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

procRTLTCust() [saimptlog_rtlog.c]

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

procRTLCAtt() [saimptlog_rtlog.c]

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

procRTLTItem() [saimptlog_rtlog.c]

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

Convert selling unit of measure to standard, necessary.

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

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

procRTLIDisc() [saimptlog_rtlog.c]

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

Convert selling unit of measure to standard, if necessary.

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

procRTLTTax() [saimptlog_rtlog.c]

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

Add the tax amount to the sale transaction total.

procRTLTTend() [saimptlog_rtlog.c]

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

Add the tender amount to the tender transaction total.

resetTran() [saimptlog_rtlog.c]

Reinitialize the transaction structures.

chkTranTailCount() [saimptlog_rtlog.c]

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

chkTranFormat() [saimptlog_rtlog.c]

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

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

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

rrchk() [saimptlog_rtlog.c]

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

WrBadTran() [saimptlog_rtlog.c]

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

reformatTran() [saimptlog_rtlog.c]

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

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

updateDataStatus() [saimptlog_datastat.pc]

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

mvSATHead() [saimptlog_manval.c]

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

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

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

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

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

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

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

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

If the transaction type is TRAT_PAIDOU:

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

Ensure that the vendor number field is not empty.

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

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

Else we do not validate.

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

Else we do not validate.

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

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

Return TRUE if all validations pass, else return FALSE.

mvSATCust() [saimptlog_manval.c]

Ensure that the customer ID has a value.

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

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

Return TRUE if all validations pass, else return FALSE.

mvSACAtt() [saimptlog_manval.c]

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

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

Return TRUE if all validations pass, else return FALSE.

mvSATItem() [saimptlog_manval.c]

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

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

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

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

Ensure that the quantity has a valid numeric value.

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

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

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

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

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

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

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

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

Return TRUE if all validations pass, else return FALSE.

mvSAIDisc() [saimptlog_manval.c]

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

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

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

Ensure that the quantity has a valid numeric value.

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

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

Return TRUE if all validations pass, else return FALSE.

mvSATTax() [saimptlog_manval.c]

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

Ensure that the tax amount has a valid numeric value.

Return TRUE if all validations pass, else return FALSE.

mvSATTend() [saimptlog_manval.c]

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

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

Ensure that the tender amount has a valid numeric value.

If the tender type group is TENT_CCARD or TENT_DCARD than:

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

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

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

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

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

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

Return TRUE if all validations pass, else return FALSE.

nextTranSeqNo() [saimptlog_nexttsn.c]

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

tdup_savedata() [saimptlog_tdup.c]

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

tdup_misstran() [saimptlog_tdup.c]

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

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

tdup_loaddata() [saimptlog_loadtdup.c]

Loads the data file of transaction number past ranges.

tdup_addtran() [saimptlog_tdup.c]

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

openSAVoucher() [saimptlog_output.c]

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

closeSAVoucher() [saimptlog_output.c]

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

writeSAVoucherFHEAD() [saimptlog_output.c]

Format and writes a FHEAD record to the voucher file.

writeSAVoucherFTAIL() [saimptlog_output.c]

Format and writes a FTAIL record to the voucher file.

writeSAVoucherData() [saimptlog_output.c]

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

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

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

(Note: it is not possible to return a voucher).

WrSoldSAVoucher() [saimptlog_output.c]

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

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

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

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

WrRedeemedSAVoucher() [saimptlog_output.c]

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

WrIssuedSAVoucher() [saimptlog_output.c]

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

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

InitOutputData() [saimptlog_sqlldr.c]

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

Open all of the temporary files for writing.

InitOutputClean() [saimptlog_sqlldr.c]

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

CreateTermRecords() [saimptlog_sqlldr.c]

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

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

WrOutputData() [saimptlog_sqlldr.c]

Writes the current transaction to the SQL*Loader files.

FinalOutputData() [saimptlog_sqlldr.c]

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

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

resetFmt() [saimptlog_sqlldr.c]

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

abortFmt() [saimptlog_sqlldr.c]

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

saveFmt() [saimptlog_sqlldr.c]

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

fmtSATranHead() [saimptlog_sqlldr.c]

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

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

fmtSACustomer() [saimptlog_sqlldr.c]

Formats a sa_customer record.

fmtSACustAttrib() [saimptlog_sqlldr.c]

Formats a sa_cust_attrib record.

fmtSATranItem() [saimptlog_sqlldr.c]

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

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

fmtSATranDisc() [saimptlog_sqlldr.c]

Formats a sa_tran_disc record.

fmtSATranTax() [saimptlog_sqlldr.c]

Formats a sa_tran_tax record.

fmtSATranTend() [saimptlog_sqlldr.c]

Formats a sa_tran_tender record.

fmtSAMissTran() [saimptlog_sqlldr.c]

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

setErrorSATHead() [saimptlog_sqlldr.c]

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

setErrorSATItem() [saimptlog_sqlldr.c]

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

setErrorSATDisc() [saimptlog_sqlldr.c]

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

setErrorSATTax() [saimptlog_sqlldr.c]

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

setErrorSATTend() [saimptlog_sqlldr.c]

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

InitOutputData() [saimptlog_insert.pc]

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

InitOutputClean() [saimptlog_insert.pc]

Frees the space allocated by InitOutputData.

CreateTermRecords() [saimptlog_insert.pc]

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

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

WrOutputData() [saimptlog_insert.pc]

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

FinalOutputData() [saimptlog_insert.pc]

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

resetFmt() [saimptlog_insert.pc]

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

abortFmt() [saimptlog_insert.pc]

Reset array indexes to the last saved transaction.

saveFmt() [saimptlog_insert.pc]

Save the array indexes for inserting the current transaction.

fmtSATranHead() [saimptlog_insert.pc]

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

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

fmtSACustomer() [saimptlog_insert.pc]

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

fmtSACustAttrib() [saimptlog_insert.pc]

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

fmtSATranItem() [saimptlog_insert.pc]

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

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

fmtSATranDisc() [saimptlog_insert.pc]

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

fmtSATranTax() [saimptlog_insert.pc]

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

fmtSATranTend() [saimptlog_insert.pc]

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

fmtSAMissTran() [saimptlog_insert.pc]

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

setErrorSATHead() [saimptlog_insert.pc]

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

setErrorSATItem() [saimptlog_insert.pc]

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

setErrorSATDisc() [saimptlog_insert.pc]

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

setErrorSATTax() [saimptlog_insert.pc]

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

setErrorSATTend() [saimptlog_insert.pc]

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

flushTranInsertArray() [saimptlog_insert.pc]

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

flushMissTranInsertArray() [saimptlog_insert.pc]

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

SAIMPTLOGFIN

main() [saimptlogfin.pc]

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

init() [saimptlogfin.pc]

retek_init should be called to initialize g_l_restart_max_counter.

Get the system options by calling fetchSaSystemOptions.

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

process() [saimptlogfin.pc]

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

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

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

Delete the terminating records, if any found.

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

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

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

final() [saimptlogfin.pc]

Call retek_close.

balanceGroupCreate() [saimptlogfin.pc]

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

nextBalGroupSeqNo() [nextbgsn.pc]

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

fixPostVoid() [saimptlogfin.pc]

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

Call TRANSACTION_SQL.CREATE_REVISIONS to create revision. Update sa_tran_head with the new revision number, setting the status to postvoided.

fixMissTran() [saimptlogfin.pc]

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

markImportDone() [saimptlogfin.pc]

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

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

Stored Procedures/Shared Modules (Maintainability)

Refer to the following documents for more details:

Package detail design 0 salock.doc

Functional Design 0 SA_misc.doc

Technical Design 0 SA_misc.doc

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

vupc_lookup	Looks up a variable PLU. Call vupc_lookup to see if it is a variable PLU. If it is a variable UPC, than set the variable parts to zero.
prom_loadfile	Loads the promotion data file generated by SAGETREF into memory.
prom_lookup	Checks that a promotion exists.
waste_loadfile	Loads the wastage data file generated by SAGETREF into memory.
waste_lookup	Looks up the wastage for an item.
code_loadfile	Loads the code type data file generated by SAGETREF into memory.
code_lookup	Checks that a code type/code exists.
error_loadfile	Loads the error data file generated by SAGETREF into memory.
error_lookup	Looks up the error and the system codes that we are interested in it.
storepos_loadfile	Loads the store POS data file generated by SAGETREF into memory.
storepos_lookup	Looks up the store POS data that we are interested in it.
tendertype_loadfile	Loads the tender type data file generated by SAGETREF into memory.
tendertype_lookup	Checks that a tender type group and ID exists.
merchcode_loadfile	Loads the merchant code data file generated by SAGETREF into memory.
merchcode_lookup	Looks up the merchant code data that we are interested in it.
partner_loadfile	Loads the partner data file generated by SAGETREF into memory.
partner_lookup	Looks up the partner data that we are interested in it.
supplier_loadfile	Loads the supplier data file generated by SAGETREF into memory.
supplier_lookup	Looks up the supplier data that we are interested in it.
putrec	Writes a record to a file.
LANGUAGE_SQL.GET_CODE_DESC	This function will retrieve the description of the passed in code and code type.

Input Specifications

The input files for Item, Wastage, sub-transaction level item (reference item), Variable PLU, Store Day, Promotions, Code Types, and Errors are all documented in the sagetref batch design.

The RTLOG file format is documented in the SA RTLog Interface.

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

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

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

Output Specifications

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

(2) call when the files are complete (see the UNIX man pages in the indicated sections for usage details).

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

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

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

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

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

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

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

Table Name	Column Name	Field Type	Field Width	Position	Description
Sa_tran_head	Tran_seq_no	Integer external	20	1:20	
	Rev_no	Integer external	3	21:23	
	Store_day_seq_no	Integer external	20	24:43	
	Tran_datetime	date	14	44:57	Format is YYYYMMDD HH24MISS
	Register	char	5	58:62	
	Tran_no	Integer external	10	63:72	
	Cashier	char	10	73:82	
	Salesperson	char	10	83:92	
	Tran_type	char	6	93:98	
	Sub_tran_type	char	6	99:104	
	Orig_tran_no	Integer external	10	105:114	
	Orig_reg_no	char	5	115:119	
	Ref_no1	char	30	120:149	
	Ref_no2	char	30	150:179	
	Ref_no3	char	30	180:209	
	Ref_no4	char	30	210:239	
	Reason_code	char	6	240:245	
	Vendor_no	char	10	246:255	
	Vendor_invc_no	char	30	256:285	
	Payment_ref_no	char	16	286:301	
	Proof_of_delivery_no	char	30	302:331	

Table Name	Column Name	Field Type	Field Width	Position	Description
	Status	char	6	332:337	
	Value	char	22	338:359	Includes an optional negative sign and a decimal point.
	Pos_tran_ind	char	1	360:360	
	Update_id	char	30	361:390	
	Update_datetime	date	14	391:404	Format is YYYYMMDD HH24MISS
	Error_ind	char	1	405:405	
Sa_tran_item	Tran_seq_no	Integer external	20	1:20	
	Item_seq_no	Integer external	4	21:24	
	Item_status	char	6	25:30	
	Item_type	char	6	31:36	
	Item	char	25	37:61	
	Ref_item	char	25	62:86	
	Non_merch_item	char	25	87:111	
	Voucher_no	char	16	112:127	
	Dept	Integer external	4	128:131	
	Class	Integer external	4	132:135	
	Subclass	Integer external	4	136:139	
	Qty	decimal external	14	140:153	Includes an optional negative sign and a decimal point.
	Unit_retail	decimal external	21	154:174	Includes a decimal point.
	Selling UOM	char	4	175:178	
	Override_reason	char	6	179:184	
	Orig_unit_retail	decimal external	21	185:205	Includes a decimal point.
	Standard_orig_unit_retail	decimal external	21	206:226	
	Tax_ind	char	1	227:227	
	Item_swiped_ind	char	1	228:228	

Table Name	Column Name	Field Type	Field Width	Position	Description
	Error_ind	char	1	229:229	
	Drop_ship_ind	char	1	230:230	
	Waste_type	char	6	231:236	
	Waste_pct	decimal external	12	237:248	Includes a decimal point.
	Pump	char	8	249:256	
	Return_reason_code	char	6	257:262	
	Salesperson	char	10	263:272	
	Expiration_date	Date	8	273:280	Format is YYYYMMDD
	Standard_qty	decimal external	14	281:294	Includes an optional negative sign and a decimal point.
	Standard_unit_retail	decimal external	21	295:315	Includes a decimal point.
	Standard_uom	char	4	316:319	
	Ref_no5	char	30	320:349	
	Ref_no6	char	30	350:379	
	Ref_no7	char	30	380:409	
	Ref_no8	char	30	410:439	
Sa_tran_disc	Tran_seq_no	Integer external	20	1:20	
	Item_seq_no	Integer external	4	21:24	
	Discount_seq_no	Integer external	4	25:28	
	rms_promo_type	char	6	29:34	
	Promotion	Integer external	10	35:44	
	Discount_type	char	6	45:50	
	Coupon_no	char	16	51:66	
	Coupon_ref_no	char	16	67:82	
	Qty	decimal external	14	83:96	Includes an optional negative sign and a decimal point.
	Unit_discount_amt	decimal external	21	97:117	Includes a decimal point.

Table Name	Column Name	Field Type	Field Width	Position	Description
	Standard_qty	decimal external	14	118:131	Includes an optional negative sign and a decimal point.
	Standard_unit_discount_amt	decimal external	21	132:152	Includes a decimal point.
	Ref_no13	char	30	153:182	
	Ref_no14	char	30	183:212	
	Ref_no15	char	30	213:242	
	Ref_no16	char	30	243:272	
	Error_ind	char	1	273:273	
Sa_tran_tax	Tran_seq_no	Integer external	20	1:20	
	Tax_code	char	6	21:26	
	Tax_seq_no	Integer external	4	27:30	
	Tax_amt	decimal external	22	31:52	Includes an optional negative sign and a decimal point.
	Error_ind	char	1	53:53	
	Ref_no17	char	30	54:83	
	Ref_no18	char	30	84:113	
	Ref_no19	char	30	114:143	
	Ref_no20	char	30	144:173	
Sa_tran_tender	Tran_seq_no	Integer external	20	1:20	
	Tender_seq_no	Integer external	4	21:24	
	Tender_type_group	char	6	25:30	
	Tender_type_id	Integer external	6	31:36	
	Tender_amt	decimal external	22	37:58	Includes an optional negative sign and a decimal point.
	Cc_no	Integer external	16	59:74	
	Cc_cc_exp_date	date	8	75:82	Format is YYYYMMDD
	Cc_auth_no	char	16	83:98	
	Cc_auth_src	char	6	99:104	

Table Name	Column Name	Field Type	Field Width	Position	Description
	Cc_entry_mode	char	6	105:110	
	Cc_cardholder_verf	char	6	111:116	
	Cc_term_id	char	5	117:121	
	Cc_spec_cond	char	6	122:127	
	Voucher_no	char	16	128:143	
	Coupon_no	char	16	144:159	
	Coupon_ref_no	char	16	160:175	
	Ref_no9	char	30	176:205	
	Ref_no10	char	30	206:235	
	Ref_no11	char	30	236:265	
	Ref_no12	char	30	266:295	
	Error_ind	char	1	296:296	
Sa_customer	Tran_seq_no	Integer external	20	1:20	
	Cust_id	char	16	21:36	
	Cust_id_type	char	6	37:42	
	Name	char	40	43:82	
	Addr1	char	40	83:122	
	Addr2	char	40	123:162	
	City	char	30	163:192	
	Sate	char	3	193:195	
	Postal_code	char	10	196:205	
	Country	char	3	206:208	
	Home_phone	char	20	209:228	
	Work_phone	char	20	229:248	
	E_mail	char	100	249:348	
	birthdate	date	8	349:356	Format is YYYYMMDD
Sa_cust_attrib	Tran_seq_no	Integer external	20	1:20	
	Attrib_seq_no	char	4	21:24	
	Attrib_type	char	6	25:30	
	Attrib_value	char	6	31:36	
Sa_error	Error_seq_no	Integer external	20	1:20	
	Store_day_seq_no	Integer external	20	21:40	

Table Name	Column Name	Field Type	Field Width	Position	Description
	Bal_group_seq_no	Integer external	20	41:60	
	Total_seq_no	Integer external	20	61:80	
	Tran_seq_no	Integer external	20	81:100	
	Error_code	char	25	101:125	
	Key_value_1	Integer external	4	126:129	
	Key_value_2	Integer external	4	130:133	
	Rec_type	char	6	134:139	
	Store_override_ind	char	1	140:140	
	Hq_override_ind	char	1	141:141	
	Update_id	char	30	142:171	
	Update_datatime	date	14	172:185	Format is YYYYMMDD HH24MISS
	Orig_value	char	50	186:235	
Sa_missing_tran	Miss_tran_seq_no	Integer external	20	1:20	
	Store_day_seq_no	Integer external	20	21:40	
	Register	char	5	41:45	
	Tran_no	Integer external	10	46:55	
	status	char	6	56:61	

Database Integrity

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

Parameter validation

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

Integrity Constraints

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

Scheduling Considerations

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

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

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

Threading Scheme

N/A

Locking Strategy

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

Restart / Recovery

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

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

Performance

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

Security Considerations

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

Appendix

CreateTermRecords

Table	Column	Value
sa_tran_head	tran_seq_no	Determined by saimptlog.
	rev_no	001
	store_day_seq_no	Same as last transaction processed.
	tran_datetime	Business Date at midnight
	tran_no	0000000000

CreateTermRecords

Table	Column	Value
	tran_type	TERM
	status	W
	pos_tran_ind	N
	ref_no1	Corresponding sa_missing_tran.miss_tran_seq_no if sa_system_options.check_dup_miss_tran = Y.
	update_id	000000000000000000000000000000
	update_datetime	SYSDATE
	error_ind	N
sa_customer	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	cust_id	0000000000000000
	cust_id_type	TERM
sa_cust_attrib	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	attrib_type	TERM
	attrib_value	TERM
sa_tran_item	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	item_seq_no	0001
	Item_status	S
	item_type	TERM
	qty	000000000000
	unit_retail_sign	P
	unit_retail	00000000000000000000
	tax_ind	N
	item_swiped_ind	N
	error_ind	N
sa_tran_disc	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	item_seq_no	0001
	rms_promo_type	TERM
	discount_seq_no	0001
	discount_type	TERM
	qty	000000000000
	unit_discount_amt_sign	P
	unit_discount_amt	00000000000000000000

CreateTermRecords		
Table	**Column**	**Value**
	error_ind	N
sa_tran_tax	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	tax_code	TERM
	tax_seq_no	0001
	tax_amt_sign	P
	tax_amt	00000000000000000000
	error_ind	N
sa_tran_tender	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	tender_seq_no	0001
	tran_type_group	TERM
	tran_type_id	000000
	tender_amt_sign	P
	tender_amt	00000000000000000000
	error_ind	N
sa_error	error_seq_no	Determined by saimptlog.
	store_day_seq_no	Same as last transaction processed.
	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	error_code	TERM_MARKER_NO_ERROR
	record_type	THEAD
	store_override_ind	N
	hq_override_ind	N
	update_id	TLOG
	update_datetime	SYSDATE
This is present only if sa_system_options.check_dup_miss_tran = Y.		
sa_missing_tran	miss_tran_seq_no	Determined by saimptlog.
	store_day_seq_no	Same as last transaction processed.
	tran_no	-000000001
	status	M

saimptlogtdup_upd (tdup File Update)

Functional Area

ReSA (Oracle Retail Sales Audit)

Module Affected

SAIMPTLOGTDUP_UPD.PC

Design Overview

The purpose of this batch module is to fetch all deleted transactions for a store day and modify the tdup<Store>.dat file to remove deleted transactions from the tdup range to facilitate saimptlog/saimptlogi batch to upload deleted transactions again. The batch will process all the store day's having data status in ('Partially Loaded' and 'Ready For Import') and business date lies between (vdate-sa_syatem_options. day_post_sale) and vdate. The batch will not process a store day, if tdup<Store>.dat file does not exist. The batch has been designed to work only if sa_system_options.check_dup_miss_tran is 'Y', otherwise do nothing and come out with successful completion.Also, the batch will not terminate with error, if the deleted transaction to be removed from tdup range doesn't exist in tdup<Store>.dat file.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Sales Audit (Daily)
Scheduling Considerations	Run towards the delete of store day or delete transaction.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

N/A

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
SA_STORE_DAY	Yes	No	No	No
SA_TRAN_HEAD	Yes	No	No	No

Shared Modules

N/A

I/O Specification

N/A