Retek

Retek Merchandising System 9.0.2.0

Addendum to Operations Guide

rms-9020-og-addendum

Retek Merchandising System™

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

Copyright Notice

Trademarks

Printed in the United States of America.

Customer Support

Customer Support hours:

8 AM to 5 PM Central Standard Time (GMT-6), Monday through Friday, excluding Retek company holidays (in 2000: Jan. 3, May 29, July 3, July 4, Sept. 4, Nov. 23, Nov. 24, Dec. 25).

Customer Support emergency hours:

24 hours a day, 7 days a week.

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

When contacting Customer Support:

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

Contents

Chapter 1 – Introduction 1

Chapter 2 – ReSA 9.0 RTLOG layout 2

Chapter 3 – saexpach batch module design 25

Functional Area 25
Module Affected 25
Design Overview 25
Background information – Quick Overview of the ACH process 26
Data Security 27
Scheduling Constraints 28
Restart Recovery 28
Program Flow 29
Shared Modules 29
Function Level Description 30
I/O Specification 33
Technical Issues 40
Assumptions 40

Chapter 4 – saimptlog batch detail design 41

Introduction 41
Functional Area 41
Module Affected 41
Design Overview 42
Program Flow 44
Function Level Description 44
SAIMPTLOGFIN 52
Stored Procedures / Shared Modules (Maintainability) 53
Input Specifications 55
Output Specifications 56
Database Integrity 60
Scheduling Considerations 60
Locking Strategy 60

Restart / Recovery 60
Performance 61
Security Considerations 61
Design Assumptions 61
Outstanding Design Issues 61
References 61
Batch Detailed Design Walkthrough 61
Appendix 62

Chapter 1 – Introduction

This addendum to the Retek Merchandising System (RMS) 9.0.0.0 Operations Guide contains updates to the following information:

- ReSA 9.0 RTLOG Layout
- saexpach batch module design
- saimptlog batch detail design

Refer to the following chapters for that information, which supercedes all comparable information in the RMS 9.0.0.0 Operations Guide.

Chapter 2 – ReSA 9.0 RTLOG layout

The following illustrates the file layout format of the Retek TLOG. The content of each Retek TLOG file is per store per day. The filename convention will be RTLOG_STORE_DATETIME.DAT (e.g. RTLOG_1234_01221989010000.DAT)

```
FHEAD                       (Only 1 per file, required)
    THEAD                   (Multiple expected, one per transaction, required for each transaction)
        TCUST               (Only 1 per THEAD record allowed, optional for some transaction types, see table below)
            CATT            (Attribute record specific to the TCUST record – Multiple allowed, only valid if TCUST exists)
        TITEM               (Multiple allowed per transaction, optional for some transaction types, see table below)
            IDISC           (Discount record specific to the TITEM record – Multiple allowed per item, optional see table below)
        TTAX                (Multiple allowed per transaction, optional see table below)
        TTEND               (Multiple allowed per transaction, optional for some transaction types, see table below)
    TTAIL                   (1 per THEAD, required)
FTAIL                       (1 per file, required)
```

The order of the records within the transaction layout above is important. It aids processing by ensuring that information is present when it is needed.

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	File Type Definition	Char(4)	RTLG	Identifies file as ‘Retek TLOG’.	Y	Left/Blank
	File Create Date	Char(14)	Create date	Date and time file was written by external system (YYYYMMDDHHMMSS).	Y	Left/None
	Business Date	Char(8)	Business Date to process	Business date of transactions. (YYYYMMDD).	Y	Left/None
	Location Number	Char(4)	Specified by external system	Store or warehouse identifier.	Y	Left/None
	Reference Number	Char(30)	Specified by external system	This may contain the Polling ID associated with the consolidated TLOG file or used for other purpose.	N	Left/Blank

Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type.	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Register	Char(5)		Till used at store.	Y	Left/Blank
	Transaction Date	Char(14)	Transaction date	Date transactions were processed at the POS (YYYYMMDDHHMMSS).	Y	Left/None
	Transaction Number	Number(10)		Transaction identifier.	Y	Right/0
	Cashier	Char(10)		Cashier identifier.	N	Left/Blank
	Salesperson	Char(10)		Salesperson identifier.	N	Left/Blank
	Transaction Type	Char(6)	Refer to 'TRAT' code_type for a list of valid types.	Transaction type.	Y	Left/Blank
	Sub-transaction type	Char(6)	Refer to 'TRAS' code_type for a list of valid types.	Sub-transaction type. For sale, it can be employee, drive-off etc.	N	Left/Blank
	Orig_tran_no	Number(10)		*Populated only for post-void transactions. Transaction number for the original tran that will be cancelled.*	*N*	*Right/0*
	Orig_reg_no	Char(5)		*Populated only for post-void transactions. Register number from the original tran.*	*N*	*Left/Blank*
	Reason Code	Char(6)	Refer to 'REAC' code_type for a list of valid codes. If the transaction type is 'PAIDOU' and the sub transaction type is 'MV' or 'EV' than the valid codes come from the non_merch_code_head table.	*Reason entered by cashier for some transaction types. Required for Paid In and Paid out transaction types, but can also be used for voids, returns, etc.*	*N*	*Left/Blank*
	Vendor Number	Char(10)		*Supplier id for a merchandise vendor paid out transaction, partner id for an expense vendor paid out transaction.*	*N*	*Left/Blank*
	Vendor Invoice Number	Char(30)		*Invoice number for a vendor paid out transaction.*	*N*	*Left/Blank*

Payment Reference Number	Char(16)		*The reference number of the tender used for a vendor payout. This could be the money order number, check number, etc.*	*N*	*Left/Blank*
Proof of Delivery Number	Char(30)		*Proof of receipt number given by the vendor at the time of delivery. This field is populated for a vendor paid out transaction.*	*N*	*Left/Blank*
Reference Number 1	*Char(30)*		Number associated with a particular transaction, for example weather for a Store Conditions transaction. The sa_reference table defines what this field can contain for each transaction type.	N	*Left/Blank*
Reference Number 2	Char(30)		Second generic reference number.	N	*Left/Blank*
Reference Number 3	Char(30)		Third generic reference number.	N	*Left/Blank*
Reference Number 4	Char(30)		Fourth generic reference number.	N	*Left/Blank*
Value Sign	Char(1)	Refer to 'SIGN' code_type for a list of valid codes.	Sign of the value.	Y if Value is present	*Left/None*
Value	Number(20)		Value with 4 implied decimal places. Populated by the retailer for TOTAL trans, populated by Retek sales audit for SALE, RETURN trans.	Y if tran is a TOTAL.	*Right/0*

Transaction Customer	File Type Record Descriptor	Char(5)	TCUST	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Customer ID	Char(16)	Customer identifier	The ID number of a customer.	Y	Left/Blank
	Customer ID type	Char(6)	Refer to 'CIDT' code_type for a list of valid types	Customer ID type.	Y	Left/Blank
	Customer Name	Char(40)		Customer name.	N	Left/Blank
	Address 1	Char(40)		Customer address.	N	Left/Blank
	Address 2	Char(40)		Additional field for customer address.	N	Left/Blank
	City	Char(30)		City.	N	Left/Blank
	State	Char(3)	State identifier	State.	N	Left/Blank
	Zip Code	Char(10)	Zip identifier	Zip code.	N	Left/Blank
	Country	Char(3)		Country.	N	Left/Blank
	Home Phone	Char(20)		Telephone number at home.	N	Left/Blank
	Work Phone	Char(20)		Telephone number at work.	N	Left/Blank
	E-mail	Char(100)		E-mail address.	N	Left/Blank
	Birthdate	Char(8)		Date of birth. (YYYYMMDD)	N	Left/Blank

Customer Attribute	File Type Record Descriptor	Char(5)	CATT	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Attribute type	Char(6)	Refer to ‘SACA' code_type for a list of valid types	Type of customer attribute	Y	Left/Blank
	Attribute value	Char(6)	Refer to members of ‘SACA' code_type for a list of valid values	Value of customer attribute.	Y	Left/Blank

Transaction Item	File Type Record Descriptor	Char(5)	TITEM	*Identifies file record type.*	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Item Status	Char(6)	Refer to 'SASI' code_type for a list of valid codes.	Status of the item within the transaction, V for item void, S for sold item, R for returned item.	Y	Left/Blank
	Item Type	Char(6)	Refer to 'SAIT' code_type for a list of valid codes.	Identifies what type of item is transmitted.	Y	Left/Blank
	SKU	Number(8)	Item identifier	ID number	Either SKU	Left/Blank
	UPC	Char(13)	Item identifier	ID number	Or UPC	Left/Blank
	Supplement	Number(5)	Supplemental identifier	Used to further specify the ID of a UPC.	N	Left/Blank
	Voucher	Char(16)		Gift certificate number	N	Right/0
	Item Number	Char(16)	Item identifier	Populated by retailer for Item types other than SKU, UPC or GCN. Allows retailers more flexibility to store additional item types within ReSA.	N	Left/Blank
	Department	Number(4)		Identifies the department this item belongs to. This is filled in by saimptlog.	N	Right/Blank
	Class	Number(4)	Item's class	Class of item sold or returned. Not required from a retailer, populated by Retek sales audit. This is filled in by saimptlog.	N	Right/Blank
	Subclass	Number(4)	Item's subclass	Subclass of item sold or returned. Not required from a retailer, populated by Retek sales audit. This is filled in by saimptlog.	N	Right/Blank
	System Indicator	Char(1)	Refer to 'IMTP' code_type for a list of valid codes.	The type of item sold or returned. Not required from a retailer, populated by Retek sales audit. This is filled in by saimptlog.	N	Left/None
	Quantity Sign	Char(1)	Refer to 'SIGN' code_type for a list of valid codes.	Sign of the quantity	Y	*Left/None*

Quantity	Number(12)		Number of items purchased with 4 decimal places.	Y	Right/0
Unit Retail	Number(20)		Unit retail with 4 implied decimal places.	Y	Right/0
Override Reason	Char(6)	Refer to 'ORRC' code_type for a list of valid codes.	This column will be populated when an item's price has been overridden at the POS to define why it was overridden.	Y if unit retail was manually entered	Left/Blank
Original Unit Retail	Number(20)		Value with 4 implied decimal places. This column will be populated when the item's price was overridden at the POS and the item's original unit retail is known.	Y if unit retail was manually entered	Right/0
Taxable Indicator	Char(1)	Refer to 'YSNO' code_type for a list of valid codes.	*Indicates whether or not item is taxable.*	Y	Left/None
Pump	*Char(8)*		Fuel pump identifier.	N	Left/Blank
Reference Number 5	*Char(30)*		Number associated with a particular item within a transaction, for example special order number. The sa_reference table defines what this field can contain for each transaction type.	N	Left/Blank
Reference Number 6	Char(30)		Second generic reference number at the item level.	N	Left/Blank
Reference Number 7	Char(30)		Third generic reference number at the item level.	N	Left/Blank
Reference Number 8	Char(30)		Fourth generic reference number at the item level.	N	Left/Blank
Item_swiped_ind	Char(1)	Refer to 'YSNO' code_type for a list of valid codes.	*Indicates if the item was automatically entered into the POS system or if it had to be manually keyed.*	*Y*	*Left/None*
Return Reason Code	Char(6)	Refer to 'SARR' code_type for a list of valid codes.	*The reason an item was returned.*	*N*	*Left/Blank*
Salesperson	Char(10)		*The salesperson who sold the item.*	*N*	*Left/Blank*
Expiration_date	Char(8)		Gift certificate expiration date (YYYYMMDD).	N	

Item Discount	File Type Record Descriptor	Char(5)	IDISC	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	RMS Promotion Number	Char(6)	Refer to 'PRMT' code_type for a list of valid types	The RMS promotion type.	Y	Left/Blank
	Discount Reference Number	Number(4)		Discount reference number is associated with the discount type (e.g. if discount type is a promotion, this contains the promotion number).	N	Left/Blank
	Discount Type	Char(6)	Refer to 'SADT' code_type for a list of valid types.	The type of discount within a promotion. This allows a retailer to further break down coupon discounts within the "In-store" promotion, for example.	N	Left/Blank
	Coupon Number	Char(16)		Number of a store coupon used as a discount.	Y if coupon	Left/Blank
	Coupon Reference Number	Char(16)		Additional information about the coupon, usually contained in a second bar code on the coupon.	Y if coupon	Left/Blank
	Quantity Sign	Char(1)	Refer to 'SIGN' code_type for a list of valid codes.	Sign of the quantity.	Y	*Left/None*
	Quantity	Number(12)		The quantity purchased that discount is applied with 4 implied decimal places.	Y	Right/0
	Unit Discount Amount	Number(20)		Unit discount amount for this item with 4 implied decimal places.	Y	Right/0
	Reference Number 13	*Char(30)*		Number associated with a particular transaction type at the discount level. The sa_reference table defines what this field can contain for each transaction type.	N	Left/Blank
	Reference Number 14	Char(30)		Second generic reference number at the discount level.	N	Left/Blank
	Reference Number 15	Char(30)		Third generic reference number at the discount level.	N	Left/Blank
	Reference Number 16	Char(30)		Fourth generic reference number at the discount level.	N	Left/Blank

Transaction Tax	File Type Record Descriptor	Char(5)	TTAX	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Tax Code	Char(6)	Refer to 'TAXC' code_type for a list of valid codes	Tax code to represent whether it is a state tax type, provincial tax, etc.	Y	Left/Blank
	Tax Sign	Char(1)	Refer to 'SIGN' code_type for a list of valid codes.	Sign of Tax Amount.	Y	*Left/None*
	Tax Amount	Number(20)		Amount of tax charged for this tax code type in a transaction with 4 implied decimal places.	Y	Right/0
	Ref_no17	Char(30)		Additional information about the tax that the retailer chooses to the store.	N	Left/Blank
	Ref_no18	Char(30)		Additional information about the tax that the retailer chooses to the store.	N	Left/Blank
	Ref_no19	Char(30)		Additional information about the tax that the retailer chooses to the store.	N	Left/Blank
	Ref_no20	Char(30)		Additional information about the tax that the retailer chooses to the store.	N	Left/Blank

Transaction Tender	File Type Record Descriptor	Char(5)	TTEND	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Tender Type Group	Char(6)	Refer to 'TENT' code_type for as list of valid types	High-level grouping of tender types.	Y	Left/Blank
	Tender Type ID	Number(6)	Refer to the pos_tender_type_head table for as list of valid types	Low-level grouping of tender types.	Y	Left/Blank
	Tender Sign	Char(1)	Refer to 'SIGN' code_type for a list of valid codes.	Sign of the value.	Y	*Left/None*
	Tender Amount	Number(20)		Amount paid with this tender in the transaction with 4 implied decimal places.	Y	Right/0
	Cc_no	Number(16)		Credit card number	Y if credit card	Left/Blank
	Cc_auth_no	Char(16)		Authorization number for a cc	Y if credit card	Left/Blank
	cc authorization source	Char(6)	Refer to 'CCAS' code_type for as list of valid types		Y if credit card	Left/Blank
	cc cardholder verification	Char(6)	Refer to 'CCVF' code_type for as list of valid types		Y if credit card	Left/Blank
	cc expiration date	Char(8)		(YYYYMMDD)	Y if credit card	Left/Blank
	cc entry mode	Char(6)	Refer to 'CCEM' code_type for as list of valid types	Indicates whether the credit card was swiped, thus automatically entered, or manually keyed.	Y if credit card	Left/Blank
	cc terminal id	Char(6)		Terminal number transaction was sent from.	N	Left/Blank
	cc special condition	Char(6)	Refer to 'CCSC' code_type for as list of valid types		Y if credit card	Left/Blank

Transaction Tender	File Type Record Descriptor	Char(5)	TTEND	Identifies file record type	Y	Left/Blank
	Voucher_no	Char(16)		Gift certificate or credit voucher serial number.	Y if voucher	Right/0
	Coupon Number	Char(16)		Number of a manufacturer's coupon used as a tender.	Y if coupon	Left/Blank
	Coupon Reference Number	Char(16)		Additional information about the coupon, usually contained in a second bar code on the coupon.	Y if coupon	Left/Blank
	Reference No 9	Char(30)		Number associated with a particular transaction type at the tender level. The sa_reference table defines what this field can contain for each transaction type.	N	Left/Blank
	Reference No 10	Char(30)		Second generic reference no at the tender level.	N	Left/Blank
	Reference No 11	Char(30)		Third generic reference no at the tender level.	N	Left/Blank
	Reference No 12	Char(30)		Fourth generic reference no at the tender level.	N	Left/Blank
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Transaction Record Counter	Number(10)		No of records processed in current tran (only records between trans head & tail)		
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	File Record Counter	Number(10)		No of transactions processed in current file (only records between file head & tail)	Y	Right/0

The RTLOG file is imported into the Sales Audit tables after validation by the batch program saimptlog. This section describes the requirements and validations performed on the records.

1. Common requirements/validations:

   This section details the common requirements and validations performed on all transactions. The following sections describe the specific requirements of each type of transaction. If a transaction is not mentioned, then it does not have specific requirements.

   a. Record Type Requirements:

Transaction Type	Includes item records?	Includes tender records?	Includes tax records?	Includes customer records?
OPEN	No	No	No	No
NOSALE	No	**Optional**	No	No
VOID	**Optional**	**Optional**	**Optional**	**Optional**
PVOID	No	No	No	No
SALE	**Yes**	**Yes**	**Optional**	**Optional**
RETURN	**Yes**	**Yes**	**Optional**	**Optional**
EEXCH	**Yes**	No	**Optional**	**Optional**
PAIDIN	No	**Yes**	No	No
PAIDOU	No	**Yes**	No	No
PULL	No	**Yes**	No	No
LOAN	No	**Yes**	No	No
COND	No	No	No	No
CLOSE	No	*No*	No	No
TOTAL	No	No	No	No
REFUND	This transaction is not sent through the RTLOG. It is entered at the HQ level. The TITEM and TCUST records are optional. The TTEND record is required. A TTAX record should not be included.			
METER	**Yes**	No	No	No
PUMPT	**Yes**	No	No	No
TANKDP	**Yes**	No	No	No
TERM	TERM records are created by saimptlog and then loaded into the database. They do not come from the RTLOG file. They require one TITEM, one TTEND, one TTAX, one TCUST record and one CATT record.			
DCLOSE	No	No	No	No

b. Requirements per record type:

Record Type	Requirements
IDISC	• IDISC records must immediately follow their associated TITEM record.
CATT	• CATT records must immediately follow their associated TCUST record.

c. Code Type Validations:

Record Name	**Field Name**	**Code Type**
Transaction Header	Transaction Type	TRAT
	Sub-transaction Type	TRAS
	Reason Code	REAC or values from non_merch_code_head if the transaction type is 'PAIDOU' and the sub transaction type is 'MV' or 'EV'.
	Value Sign	SIGN
	Vender No	If the transaction type is 'PAIDOU' and the sub transaction type is 'MV', this field is validated against the supplier table. If the transaction type is 'PAIDOU' and the sub transaction type is 'EV', this field is validated against the partner table.
Transaction Item	Item Type	SAIT
	Item Status	SASI
	System Indicator	IMTP
	Quantity Sign	SIGN
	Taxable Indicator	YSNO
	Price Override Reason Code	ORRC
	Item Swiped Indicator	YSNO
	Return Reason Code	SARR
Item Discount	RMS Promotion Type	PRMT
	Discount Type	SADT
	Quantity Sign	SIGN
Transaction Customer	Customer ID Type	CIDT
Customer Attribute	Attribute Type	SACA
	Attribute value	Code types from codes in SACA.
Transaction Tax	Tax code	TAXC
	Tax sign	SIGN
Transaction Tender	Tender Type Group	TENT
	Tender Sign	SIGN
	Tender Type ID	Pos_tender_type_head table

	CC Authorization Source	CCAS
	CC Cardholder Verification	CCVF
	CC Entry Mode	CCEM
	CC Special Condition	CCSC

d. Dates are validated: Business Date, Transaction Date, Expiration Date Also, saimptlog accepts only business dates that are within the PERIOD.VDATE minus the SA_SYSTEM_OPTIONS.DAYS_POST_SALE value.

e. Store number is validated against the STORE table.

f. Numeric fields are checked for non-numeric characters.

g. For transaction of type SALE, RETURN and EEXCH, saimptlog checks whether a transaction is in balance:
Transaction Items (Unit Retail * Unit Retail Sign * Quantity)
+ Item Discounts (Unit Discount Amount * Unit Discount Sign * Quantity)
+ Transaction Tax (Tax Amount * Tax Sign)
= Transaction Tenders (Tender Amount * Tender Sign)
saimptlog will populate the Value field (on THEAD) with the transaction's sales value (item value – discount value + tax value) from the above calculation if it was not provided in the RTLOG.

h. Treatment of vouchers.
   I. If an item sold is a gift certificate (Transaction Item, Voucher field has a value), issued information is written to the SA_VOUCHER table.
   II. If the Transaction Type is a RETURN, and the Transaction Tender Type Group is voucher (VOUCH), issued information is written to the SA_VOUCHER table.
   III. If the Transaction Type is a SALE, and the Transaction Tender Type Group is a voucher (VOUCH), redeemed information is written to the SA_VOUCHER table.
   IV. When a gift certificate is sold, customer information should always be included. A receiving customer name value should be populated in the ref_no5 field, a receiving customer state value should be populated in the ref_no6 field and a receiving customer country should be populated in the ref_no7 field. These reference fields can be changed by updating the sa_reference table *but the code needs to be modified too*. The expiration date is put on the expiration_date field on the TITEM record.

i. Other validations/points of interest:
   I. A salesperson in the TITEM record takes precedence over the salesperson in the THEAD record.
   II. If an item sold is a UPC (Transaction Item, UPC field has a value and SKU does not), it will be converted to the corresponding SKU using the Supplement.
   III. If an item sold is a SKU (Transaction Item, SKU field has a value), it will be validated against RMS item tables.

IV. The corresponding Department, Class, Subclass, System Indicator and Taxable Indicator will be selected from the RMS tables and populated for a SKU.

j. The balancing level determines whether the register or the cashier fields are required.
   I. If the balancing level is 'R'egister, then the register field on the THEAD must be populated.
   II. If the balancing level is 'C'ashier, then the cashier field on the THEAD must be populated.
   III. If the balancing level is 'S'tore, then neither field is required to be populated.

k. The tax_ind and the item_swiped_ind fields can only accept 'Y' or 'N' values. If an invalid value is passed through the RTLOG, an error will be flagged and the value will be defaulted to 'Y'.

2. Transaction of type 'SALE':

A transaction of type SALE is generated whenever an item is sold. A sale may be to an employee, the sub-transaction type would be EMP in this case. Or it may be a drive-off sale (sub-transaction type DRIVEO) when someone drives off with unpaid gas. A special type of sale is an "odd exchange" (sub-transaction type EXCH) where items are sold and returned in the same transaction. If the net value of the exchange is positive, then it is a sale. If the net value is negative, it is a return. If the net value is zero and the items exchanged are in the same SKU style, it would be a transaction of type EEXCH (Even Exchange).

a. Requirements per record type (other than what is described in Layout section above):

Record Type	Requirements
THEAD	
TITEM	• Item Status is a required field; it determines whether the item is 'S'old, 'R'eturned or 'V'oided. If the item status is S, the quantity sign is expected to be P. If the item status is 'R', the quantity sign is expected to be N. • If the item status is V, the quantity sign is the reverse of the quantity sign of the voided item. That is, if an item with status S is voided, the quantity sign would be N. Furthermore, the sum of the quantities being voided cannot exceed the sum of the quantities 'S'old or 'R'eturned. Note: neither of the above two validations are performed by saimptlog but an audit rule could be created to check this. • In a typical sale, the items would all have a status of 'S'. In the case of an odd exchange, some items will have a status of 'R'. • In a typical return, the items would all have a status of 'R'. In the case of an odd exchange, some items will have a

	status of 'S'. • If an item has status R, then the Return Reason Code field may be populated. If it is, it will be validated against code type 'SARR'. • If the price of an item is overridden, then the Override Reason and Original Unit Retail fields must be populated.
IDISC	• The RMS Promotion Type field must always be populated with values of code type 'PRMT'. • The Promotion field is validated, when a value is passed, against the promhead table. • If the promotion is 'In Store' (code 1004), then the Discount Type field must be populated with values of code type 'SADT'. • The Discount Reference Number is a promotion number which is of status 'A', 'E' or 'M'. • If the Discount Type is 'SCOUP' for Store Coupon, then the Coupon Number field must be populated. The Coupon Reference Number field is optional.
TTEND	• If the tender type group is 'COUPON', then the Coupon Number field must be populated. The Coupon Reference Number field is optional. • If the Transaction Tender Type Group is a credit card (CCARD), the number will be validated against the SA_CC_VAL table. The other cc fields are optional.

b. Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Transaction Type	Sub-transaction Type	Item Type	Tender Type Group	Reference Number Field	Meaning of Reference Field	Req ?
SALE				1	Speed Sale Number	Y
SALE		GCN		5	Recipient Name	N
SALE		GCN		6	Recipient State	N

SALE		GCN		7	Recipient Country	N
SALE			CHECK	9	Check Number	N
SALE			CHECK	10	Driver's License Number	N
SALE			CHECK	11	Credit Card Number	N
SALE	DRIVEO			1	Incident Number	Y
SALE	EMP			3	Employee Number of the employee receiving the goods.	N

c. Expected values for sign fields

TRANSACTION TYPE	**TITEM.Quantity Sign**	**TTEND.Tender Sign**	**TTAX.Tax Sign**	**IDISC.Quantity Sign**
SALE	P if item is sold; N if item is returned; reverse of original item if item is voided.	*P*	P	P if item is sold; N if item is returned; reverse of original item if item is voided.

3. Transaction of type 'PVOID':

   This transaction is generated at the register when another transaction is being post voided. The orig_tran_no and orig_reg_no fields must be populated with the appropriate information for the transaction being post voided. The PVOID transaction must be associated with the same store day as the original transaction. If the PVOID needs to be generated after the store day is closed, the transaction needs to be created using the forms.

4. Transaction of type 'RETURN':

   This transaction is generated when a customer returns an item.

   a. This type of transaction has similar record type requirements as a 'SALE' transaction.

   b. Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Transaction Type	Sub-transaction Type	Reference Number Field	Meaning of Reference Field	Req?
RETURN		1	Receipt Indicator (Y/N)	*Y*
RETURN		2	Refund Reference Number	N
RETURN	EMP	3	Employee Number of the employee returning the goods.	N

c. Expected values for sign fields

TRANSACTION TYPE	TITEM.Quantity Sign	TTEND.Tender Sign	TTAX.Tax Sign	IDISC.Quantity Sign
RETURN	P if item is sold; N if item is returned; reverse of original item if item is voided.	N	N	P if item is sold; N if item is returned; reverse of original item if item is voided.

5. Transaction of type 'EEXCH':

This transaction is generated when there is an even exchange.

a. This type of transaction has similar record type requirements as a 'SALE' transaction.

b. It is expected that the number of items returned equals the number of items sold. However, this validation is not performed by saimptlog. An audit rule could be created for this. Saimptlog only expects that there would be at least two item records.

c. No tender changes hands in this transaction.

d. Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Transaction Type	Sub-transaction Type	Reference Number Field	Meaning of Reference Field	Req?
EEXCH		1	Receipt Indicator (Y/N)	Y
EEXCH	EMP	3	Employee Number of the employee exchanging the goods.	N

6. Transaction of type 'PAIDIN':

   a. This type of transaction has only one TTEND record.

   b. A reason code is required.

   c. Meaning of reference number fields:

   Note: The meaning of these reference number fields may be changed through the sa_reference table.

Reason Code	Reference Number Column	Meaning	Req?
NSF	1	NFS Check Credit Number	N
ACCT	1	Account Number	N

7. Transaction of type 'PAIDOU':

   a. This type of transaction has only one TTEND record.

   b. A reason code is required (code type REAC). If the sub-transaction type is 'EV' or 'MV', the reason code comes from the non_merch_codes_head table.

c. If the sub-transaction type is 'EV' or 'MV', then at least one field among the vendor number, vendor invoice number, payment reference number and proof of delivery number fields should be populated.

d. If the sub-transaction type is 'EV', then the vendor number comes from the partner table. If the sub-transaction type is 'MV', then the vendor number comes from the supplier table.

e. Meaning of reference number fields:

Notes: The meaning of these reference number fields may be changed through the sa_reference table.

Sub Transaction Type	Reason Code	Reference Number Column	Meaning	Req?
EV		2	Personal ID Number	N
EV		3	Routing Number	N
EV		4	Account Number	N
	PAYRL	1	Money Order Number	N
	PAYRL	2	Employee Number	N
	INC	1	Incident Number	N

8. Transaction of type 'PULL':

This transaction is generated when cash is withdrawn from the register.

a. This type of transaction has only one TTEND record.

b. Expected values for sign fields

TRANSACTION TYPE	**TITEM.Quantity Sign**	**TTEND.Tender Sign**	**TTAX.Tax Sign**	**IDISC.Quantity Sign**
PULL	N/A	N	N/A	N/A

9. Transaction of type 'LOAN':

This transaction is generated when cash is added to the register.

a. This type of transaction has only one TTEND record.

b. Expected values for sign fields

TRANSACTION TYPE	**TITEM.Quantity Sign**	**TTEND.Tender Sign**	**TTAX.Tax Sign**	**IDISC.Quantity Sign**
LOAN	N/A	P	N/A	N/A

10. Transaction of type 'COND':

This transaction records the condition at the store when it opens. There can be at most one COND record containing weather information and at most one COND record containing temperature information. Both these pieces of information may be in the same COND record. There may be any number of COND records containing traffic and construction information.

a. This type of transaction does not have TITEM or IDISC or TTAX or TTEND records.

b. Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Reference Number Column	Meaning	Req?
1	Weather – code type 'WEAT'	N
2	Temperature – a signed 3 digit number.	N
3	Traffic – code_type 'TRAF'	N
4	Construction – code_type 'CONS'	N

11. Transaction of type 'TOTAL':

    This transaction records the totals that are reported by the POS. The value field must be populated. Some POS systems generate only one transaction number for all totals. In order to avoid duplicate errors to be reported, only one total transaction can have a transaction number and the subsequent ones can have blank transaction numbers. In other words, a TOTAL transaction is not required to have a transaction number.

    a. This type of transaction does not have TITEM or IDISC or TTAX or TTEND records.

12. Transaction of type 'METER':

    This transaction is generated when a meter reading of a fuel pump is taken.

    a. This type of transaction has only TITEM records.

b. Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Reference Number Column	**Meaning**	**Req?**
1	Reading Type: (‘A’ Adjustment, ‘S’ shift change, ‘P’ price change, ‘C’ store close)	Y
5	Opening Meter Readings	Y
6	Closing Meter Reading	Y
7	If the reading type is ‘P’ for price change, the old unit retail should be placed here. Decimal places are required.	Y
8	Closing Meter Value	Y

13. Transaction of type ‘PUMPT’:

This transaction is generated when a pump test is performed. This type of transaction has only TITEM records.

14. Transactions of type ‘TANKDP’:

This transaction is generated when a tank dip measurement is taken.

a. This type of transaction has only TITEM records.

b. Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Reference Number Column	Meaning	Req?
1	Tank identifier	Y
5	Dip Type ('FUEL', 'WATER', etc.)	Y
6	Dip Height Major (decimal places required)	Y
7	Dip Height Minor (decimal places required)	Y

15. Transaction of type 'DCLOSE':

This transaction is generated when day closed. Transaction number for this type of transaction has to be blank.

16. A note about vouchers: Vouchers are minimally handled by saimptlog. Voucher information is written to the savouch file which is passed to the program savouch.pc. For more information about this interface, see Interface File – SA Vouch and Batch Design – savouch.

A voucher will appear on the TITEM record only if it was sold. Thus when saimptlog encounters a 'SALE' transaction with a voucher, it writes the voucher to the savouch file as an 'I'ssued voucher.

A voucher will be issued when it appears on the TTEND record of transactions of type 'RETURN' and 'PAIDOU'. In other words, saimptlog will write it to the savouch file with status 'I'.

A voucher will be redeemed when it appears on the TTEND record of transactions of type 'SALE' and 'PAIDIN'. In other words, saimptlog will write it to the savouch file with status 'R'.

Vouchers may not be returned. However, a transaction of type 'PAIDOU' may be generated when the customer exchanges a voucher for another form of tender.

Chapter 3 – saexpach batch module design

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.

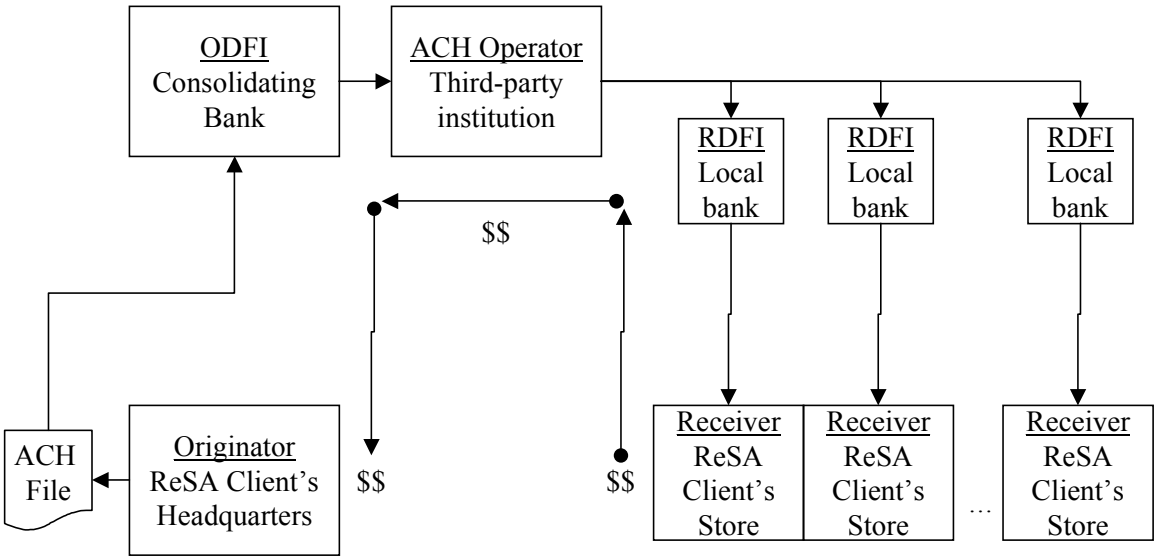


Figure 1: Overview of an ACH Network

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 Retek assumes that the ACH network is secure.

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Anytime – Sales Audit 9.0 is a 24/7 system.

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

Threading Scheme: N/A

Restart Recovery

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()**.
   - If the amount to be transferred is negative, the record should be skipped.
   - If the account is a checking account, the transaction code to use is '27'.
   - If the account is a savings account, the transaction code to use is '37'.
8. If the amount to be transferred is positive, call the function **PostBankACHTable()** to record the amount of the ACH entry, else do nothing.
9. Keep running totals for the current batch's total amount and the total ACH amounts.
10. Commit after pl_commit_max_ctr LUW have been processed. Redefine the SAVEPOINT after the commit because savepoints are lost after a commit.
11. Loop to step 3 until the cursor returns no data.
12. Write the ACH Batch Control record and the ACH File Control record
13. The ACH file format requires that the file size meet certain "block" requirements. See the section on the ACH file format for more details. Write the required number of "completion records" to meet the blocking requirements.
14. Mark all store/days *that were not locked* (i.e. those with a sa_export_log.status of SAES_P) as completed (SAES_E) in the sa_export_log.

postBankACHTable ()

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

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

File 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

All files layouts input and output

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	SA_BANK_STORE. CONSOLIDATING_ROU TING_NO	10	None

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

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

Batch Header Record for CCD transactions

Field Name	Field Description	Value	Length	Jstf/ Pad*
Record Type Code	The type of record.	'5'	1	None
Service Class Code	*This field identifies the general classification of dollar entries to be exchanged. Funds will always flow from the local banks to the consolidating bank. Hence the code '225' for "ACH Debits only" should be used.*	*'225'*	3	None
Company Name	The name of the company.	First 16 characters of COMPHEAD. CO_NAME	16	L/B
Company Discretionary Data	Any kind of data specific to the company. ReSA will not use this field	*blanks*	20	None
Company Identification	An alphanumeric code identifying the company. The first character may be the ANSI one-digit Identification Code Designators (ICD). For example, "1" IRS Employer ID Number "9" User Assigned Number. *ReSA assumes that the company_id field on the sa_system_options table will contain the correct id.*	SA_SYSTEM_OPTIONS. COMPANY_ID	10	L/B
Standard Entry Class Code	This provides a way to distinguish between the various kinds of entries. Since ReSA will be sending CCD	'CCD'	3	None

Field Name	Field Description	Value	Length	Jstf/ Pad*
	entries, this field should hold the value 'CCD'.			
Company Entry Description	A short description from the Originator about the purpose of the entry.	'CONSOL.'	10	L/B
Company Descriptive Date	Optional field providing a date to the Receiver for descriptive purposes. ReSA will populate it with the next day's date in the YYMMDD format.	YYMMDD format of PERIOD.VDATE + 1	6	None
Effective Entry Date	*The date by which the Originator intends the batch of entries to be settled. Since the Originator will want this to be done as soon as possible, ReSA will use the earliest possible date, which is one banking day after the processing date (the current date).*	YYMMDD format of PERIOD.VDATE + 1	6	None
Settlement Date	This is inserted by receiving ACH Operator. ReSA will leave this blank.	blanks	3	None
Originator Status Code	This field stores a code to describe the type of Originator. This should be a 1 to describe the Originator as a depository financial institution.	'1'	1	None
ODFI Identification	8-digit routing number of the ODFI.	First 8 digits of SA_BANK_STORE. CONSOLIDATING_ROUTING_NO	8	None
Batch Number	*The batch number.*		7	R/0

Batch Header Record for CBR transactions

Field Name	Field Description	Value	Length	Jstf/ Pad*
Record Type Code	The type of record.	'5'	1	None
Service Class Code	*This field identifies the general classification of dollar entries to be exchanged. Funds will always flow from the local banks to the consolidating bank. Hence the code '225' for "ACH Debits only" should be used.*	*'225'*	3	None
Company Name	The name of the company.	First 16 characters of COMPHEAD. CO_NAME	16	L/B
Foreign Exchange Indicator	Code used to indicate the foreign exchange conversion methodology applied to a CBR entry. Retek uses the "Fixed-to-Variable" method to convert from the foreign currency into US dollars. Therefore, this field should be 'FV'.	*'FV'*	2	None
Foreign Exchange Reference Indicator	Code used to indicate the contents of the Foreign Exchange Reference field. The latter will contain the conversion rate used by Retek which means that the value should be '1'.	*'1'*	1	None
Foreign Exchange Reference	This should contain the foreign exchange rate used to compute the amounts in the CBR Entry Record. No decimal places are implied, that is, this field should contain the exact rate used.		15	L/B
ISO Destination Country Code	The country where the money is to be transferred to. Since ReSA assumes that the consolidating bank will be in the US, this should be 'US' – NOTE: verify that "US" is the correct ISO code for United States of America.	*'US'*	2	None
Company Identification	An alphanumeric code identifying the company. The first character may be the ANSI one-digit Identification Code	SA_SYSTEM_OPTIONS. COMPANY_ID	10	L/B

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

CCD Entry Detail Record

Field Name	Field Description	Value	Length	Jstf/ Pad*
Record Type Code	The type of record.	'6'	1	None
Transaction Code	Code used to identify the type of debit and credit. This is dependent on the type of account and on the direction of funds transfer. '27' – if the account is a checking account, '37' – if the account is a savings account.	*'27' or '37'*	2	None
RDFI Identification	8-digit routing number of the RDFI.	First 8 digits of SA_BANK_STORE. ROUTING_NO	8	None
Check Digit	This is the 9th digit from the routing number.	9th digit of SA_BANK_STORE. ROUTING_NO	1	None
DFI Account Number	The account number at the local bank.	SA_BANK_STORE. BANK_ACCT_NO	17	L/B
Amount	The amount involved in the transaction. This field is numeric only and the last two digits are automatically assumed to be decimals. ReSA amounts are stored as 20 digit numbers, with 4 for decimals. ReSA will truncate the last two digits of the amount and should the resulting		10	R/0

Field Name	Field Description	Value	Length	Jstf/ Pad*
	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.			
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. ReSA has no use for such records in CCD and will use the value of '0'	'0'	1	None
Trace Number	*Used to uniquely identify each entry within a batch. The first 8 digits contain the routing number of the ODFI and the other 7 contains a sequence number. This sequence number should be ascending. Although the ACH specification does not require the numbers to be consecutive, ReSA will use consecutive numbers. Trace numbers should not be duplicated between batches.*		15	None

CBR Entry Detail Record

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

Field Name	Field Description	Value	Length	Jstf/ Pad*
	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.			

CBR Addendum Record

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

Batch Control Record

Field Name	Field Description	Value	Length	Jstf/ Pad*
Record Type Code	The type of record	'8'	1	None
Service Class Code	This field identifies the general classification of dollar entries to be exchanged. Since money is being requested, this code should be 225 for "ACH Debits only".	*'225'*	3	None
Entry/Addenda Count	The number of entries and addenda in the batch. Basically, this is the number of records between the Batch Header Record and the Batch Control Record.		6	R/0
Entry Hash	This is the sum of the RDFI IDs in the detail records. It is the arithmetic sum of the 8-digit routing number. Overflow on the high order bits is ignored.		10	R/0
Total Debit Entry Dollar Amount in batch	These fields contain the accumulated debit and credit for the batch. This field is numeric only and the last two digits are automatically assumed to be decimals.		12	R/0
Total Credit Entry Dollar Amount in batch			12	R/0
Company Identification	An alphanumeric code identifying the company. The first character may be the ANSI one-digit Identification Code Designators (ICD). For example, "1" IRS Employer ID Number	SA_SYSTEM_OPTIONS. COMPANY_ID	10	L/B

Field Name	Field Description	Value	Length	Jstf/ Pad*
	"9" User Assigned Number. ReSA assumes that the company_id field on the sa_system_options table will contain the correct id.			
Message Authentication Code (MAC)	The first 8 characters represent a code from the DES (Data Encryption Standard) algorithm. The remaining eleven characters are blanks. ReSA will not populate this field.	blanks	19	None
Reserved	Reserved	blanks	6	None
ODFI Identification	8-digit routing number of the ODFI.	First 8 digits of SA_BANK_STORE. CONSOLIDATING_ROUTING_NO	8	None
Batch Number	*The batch number.*		7	R/0

File Control Record

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

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

Technical Issues

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

Assumptions

1. This document assumes that the tables and forms changes are going to be applied accordingly.
2. It is assumed that the consolidating bank is US-based.
3. ReSA will assume that all country codes and all currency codes are ISO compliant.

Chapter 4 – saimptlog batch detail design

Document Revision History			
Revision #	**Date**	**Author**	**Brief Revision Description**
Revision 1	1/13/00	Chuck Rudolph	Phase 1
Revision 1.1	1/18/00	Chuck Rudolph	Modify saimptlog to input new fields into the Sales Audit tables.

Please note that the revision number should match the document file name.

The Project Manager must determine the need to revise this document and frequency. It may be decided that once the final walkthrough and approvals are made, only subsequent documentation will maintain current project information (such as design documents, etc.) rather than any changes being made to this document. This document, would, in essence, be a 'snapshot in time' of the project without revision following approval. The Project Manager may; however, call for this document's update originating from significant scope (objective or requirement) changes. The determined method should be documented here.

I. Introduction

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.

*A Batch Detailed Design should **not** include code (SQL).*

II. Functional Area

Sales Audit import.

III. Module Affected

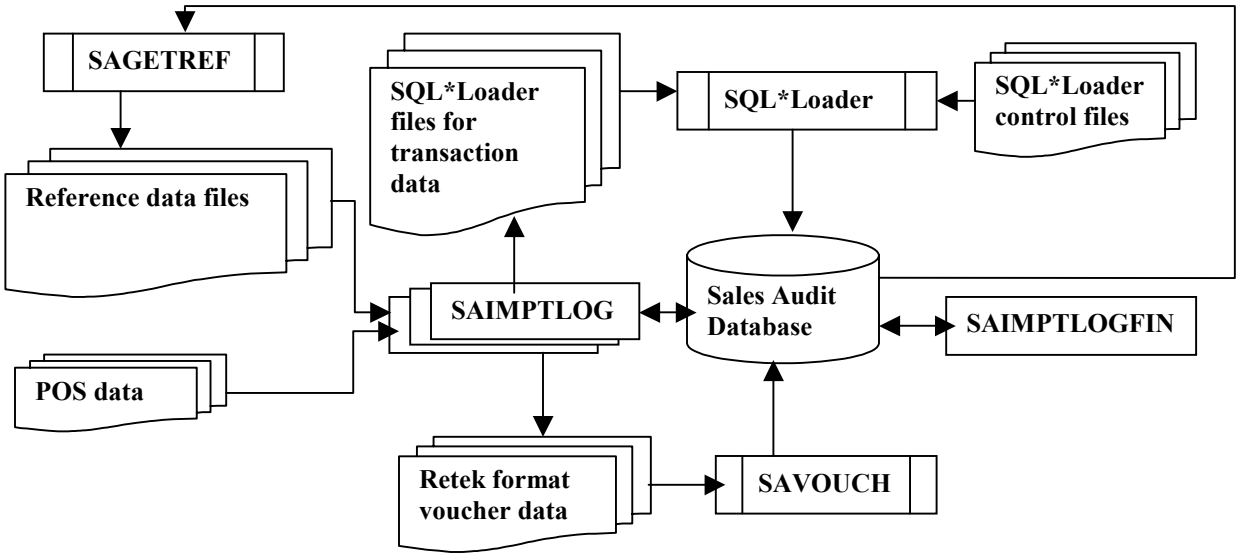
SAIMPTLOG (formerly saval.pc and saout.pc in 8.X)

saimptlog.c
saimptlog.h
saimptlog_final.c
saimptlog_init.c
saimptlog_manval.c
saimptlog_nexttsn.pc
saimptlog_nextvhn.pc
saimptlog_output.c

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

SAIMPTLOGFIN
saimptlogfin.pc
saimptlog_nexttbgsn.pc
saimptlog.h

IV. Design Overview



Design Overview

Importing POS data is a five-step process.

First, SAGETREF must be run to generate the current reference files:

- SKU
- Wastage
- UPC
- variable weight UPC
- 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. Since SAIMPTLOG can be threaded, this boosts performance by limiting interaction with the database.

Second, SAIMPTLOG is run against each POS file. SAIMPTLOG creates a write lock for store/day that is held 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 the processing by SAVOUCH. SAIMPTLOG may be threaded as long as the parallel executions do not include the same store/day.

Third, 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.

Fourth, SAVOUCH is executed to load each of the Retek formatted voucher files. SAVOUCH may not be multiply threaded.

Fifth, SAIMPTLOGFIN is executed to populate the sa_balance_group table, to mark the import as either partially or fully complete, and to release the store/day write lock that was established by SAIMPTLOG. SAIMPTLOGFIN may not be multiply threaded.

This design document encompasses SAIMPTLOG 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_data	yes	no	no	no
sa_store_day	yes	yes	no	no
sa_store_day_write_lock	yes	yes	no	no
sa_import_log	yes	yes	no	no
sa_export_log	no	yes	no	no

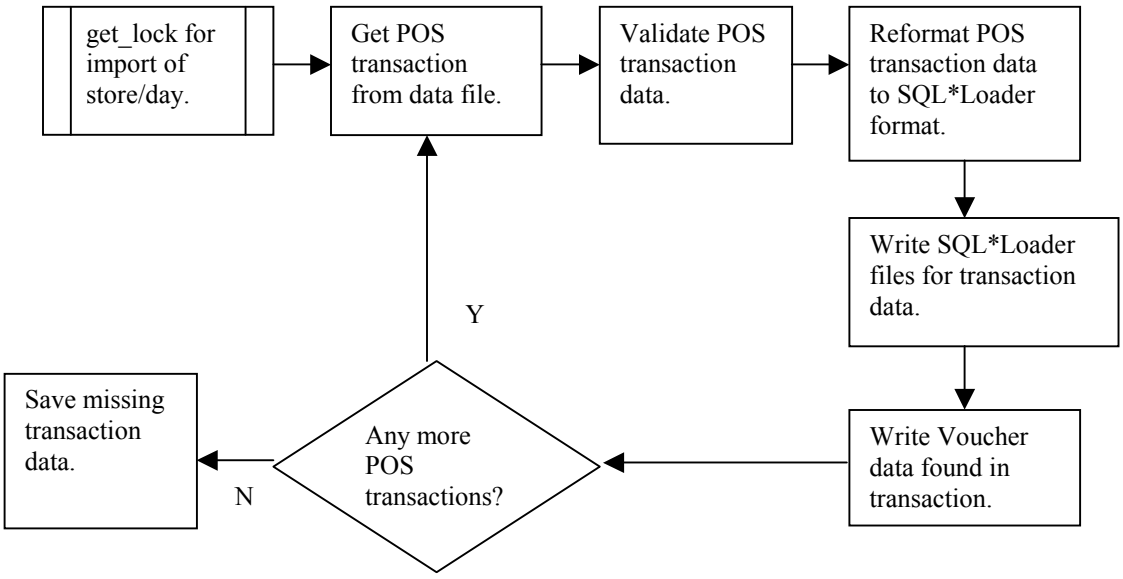
SAIMPTLOGFIN

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	no	yes
sa_import_log	yes	no	yes	no

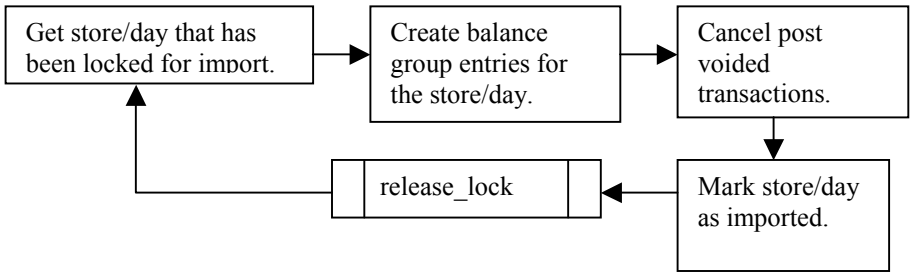
saimptlog.c is a combination of 2 programs from Sales Audit 2.0: saval.pc and saout.pc. For details on these 2 programs, refer to:
Batch Design – saval.doc
Batch Design – saout.doc
The source for these 2 programs can be found in PVCS. See Project *RMS 8.1* and Folders *Batch – Sales Audit 2.0* and *Batch Library – SA*.

V. Program Flow

SAIMPTLOG



SAIMTLOGFIN



VI. Function Level Description

SAIMPTLOG

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 the SQL*Loader and Retek formatted files. 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.
For each transaction, call **MandatoryValidations** to validate the data and than call **WrOutputData** to write the transaction to the temporary files.

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 SKU data generated by SAGETREF by calling **sku_loadfile**.
Load the UPC data generated by SAGETREF by calling **upc_loadfile**.
Load the variable weight UPC data generated by SAGETREF by calling **vupc_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.
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. Also generate a temporary filename for the voucher data.
Open all of the temporary files for writing.

Final() [saimptlog_final.c]

Call **CreateTermRecords** to mark the end of the data and than call **WrOutputData** to write them to the temporary files.
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 and call **tdup_misstran** to create the SQL*Loader file for the sa_missing_tran table.
Terminate the RTLOG file parser by calling **FinalInputData**.
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).
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.
FTAIL - Write out a FTAIL record to the voucher file.
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.

MandatoryValidations() [saimptlog_manval.c]

For each THEAD, TCUST, CATT, TITEM, IDISC, TTAX and TTEND record in the transaction, call **mvSATHead**, **mvSATCust**, **mvSACAtt**, **mvSATItem**, **mvSAIDisc**, **mvSATTax** and **mvSATTend** to make sure that the current transaction does not have non-numeric data in number fields, an invalid date in a date field, etc.

mvSATHead() [saimptlog_manval.c]

Ensure that the transaction date and time has a valid value.
Ensure that, if they exist, the cashier and salesperson ids are valid by calling employee_lookup.
Ensure that, if the balancing level is 'R', then the register field is populated, and that if the balancing level is 'C', then the cashier field is populated.
Ensure that the transaction type has a valid value (code_type of TRAT) by calling code_lookup.
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 '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.

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

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.

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.

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 'S', 'R', and 'V'.
Ensure that the item type has a valid value (code_type of SAIT) by calling code_lookup.

Ensure that the SKU, UPC, UPC supplement, voucher number or item 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 is 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, the salesperson id is valid by calling employee_lookup.
Ensure that if an expiration date exists, that it is valid.

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**.
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.

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.

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 Credit Card then:

- 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. Also, 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.

CreateTermRecords() [saimptlog_rtlog.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 following table. All other columns should be blank.

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

Table	Column	Value
	ref_no1	Corresponding sa_missing_tran.miss_tran_seq_no if sa_system_options.check_dup_miss_tran = Y.
	update_id	00000000000000000000000000000000
	update_datetime	SYSDATE
	error_ind	N
sa_customer	*tran_seq_no*	Same as sa_tran_head.tran_seq_no.
	cust_id	0000000000000000
	cust_id_type	TERM
sa_cust_attrib	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	attrib_type	TERM
	attrib_value	TERM
sa_tran_item	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	item_seq_no	0001
	Item_status	S
	item_type	TERM
	qty	000000000000
	unit_retail_sign	P
	unit_retail	00000000000000000000
	tax_ind	N
	item_swiped_ind	N
	error_ind	N
	var_upc_ind	N
sa_tran_disc	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	item_seq_no	0001
	rms_promo_type	TERM
	discount_seq_no	0001
	discount_type	TERM
	qty	000000000000
	unit_discount_amt_sign	P
	unit_discount_amt	00000000000000000000
	error_ind	N
sa_tran_tax	tran_seq_no	Same as sa_tran_head.tran_seq_no.
	tax_code	TERM
	tax_seq_no	0001
	tax_amt_sign	P
	tax_amt	00000000000000000000
	error_ind	N
	Ref_no17	
	Ref_no18	
	Ref_no19	
	Ref_no20	
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

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

WrOutputData() [saimptlog_output.c]
Writes the current transaction to the SQL*Loader files.
If the current transaction type is a sale (SALE), or a return (RETURN) and the TITEM records contains 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).

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. Call **tdup_loaddata** to load into memory past transaction number ranges for the current store/day.

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.
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.
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 produces 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 an error log file. These transactions do not contain enough information to be loaded to the Sales Audit tables for correction by an auditor.

reformatTran() [saimptlog_rtlog.c]
Reformat the data within the transaction into the SQL*Loader flat file format. This is accomplished by calling routines that know the format for each tables SQL*Loader control file. These routines are **fmtSATranHead**, **fmtSACustomer**, **fmtSACustAttrib**, **fmtSATranItem**, **fmtSATranDisc**, **fmtSATranTax** and **fmtSATranTend**.
If the transaction type is 'SALE', 'RETURN' or 'EEXCH', than check that the transaction balances by comparing the sale and tender transaction totals. Generate an error if they do not match.

fmtSATranHead() [saimptlog_rtlog.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_rtlog.c]
Formats a sa_customer record.

fmtSACustAttrib() [saimptlog_rtlog.c]
Formats a sa_cust_attrib record.

fmtSATranItem() [saimptlog_rtlog.c]
Formats a sa_tran_item record. If the item contains a variable weight UPC, than call **waste_lookup** to get the wastage type and percent. If the type is an UPC, it will be converted to a SKU. The merchandise hierarchy information (department, class, sub-class, and system indicator) associated with the SKU will be retrieved for this item by calling **sku_lookup**.
Produce an error if the SKU cannot be found, the UPC was not converted to an SKU, the item type is not SKU, UPC or GCN, or non-numeric data is found in the quantity or amount field.

fmtSATranDisc() [saimptlog_rtlog.c]
Formats a sa_tran_disc record.

fmtSATranTax() [saimptlog_rtlog.c]
Formats a sa_tran_tax record.

fmtSATranTend() [saimptlog_rtlog.c]
Formats a sa_tran_tender record.

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.

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 in SQL*Loader format.
The sa_missing_tran.status column will be filled in with SAMS_M.

tdup_loaddata() [saimptlog_tdup.c]
Loads the data file of transaction number past ranges.

tdup_addtran() [saimptlog_tdup.c]
Adds a transaction number to the list of numbers encountered. If store.tran_no_generated is SRTG_S, than the transaction number must be unique to the store. If store.tran_no_generated is SRTG_R, than the transaction number must be unique to the store and register.

VII. 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, and to release the store/day write lock that was established by SAIMPTLOG. 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 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.
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.

fixMissTran() [saimptlogfin.pc]
Remove sa_missing_tran records that may now be present because data was processed out of order.

markImportDone() [saimptlogfin.pc]
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 (CLOSE).
If there was a CLOSE transaction, than update the sa_import_log table's status and datetime columns. If the import was expected, than set status to loaded (L), else set it to unexpected (U). This is determined by calling **storeday_lookup**.

VIII. Stored Procedures / Shared Modules (Maintainability)

Refer to the following documents for more details:
Package detail design - salock.doc
Functional Design - SA_misc.doc
Technical Design - SA_misc.doc

Shared Module	*Module Description*
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.
sku_loadfile	Loads the SKU data file generated by SAGETREF into memory.
sku_lookup	Looks up a SKU and returns the data (department, class, sub-class and system indicator) associated with it.
upc_loadfile	Loads the UPC data file generated by SAGETREF into memory.
upc_lookup	Looks up a UPC.
vupc_loadfile	Loads the variable weight UPC data file generated by SAGETREF into memory.
`vupc_lookup	Looks up a variable UPC. Call **vupc_lookup** to see if it is a variable UPC. 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.

Shared Module	Module Description
waste_lookup	Looks up the wastage for a SKU.
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_loadfileMerchcode_loadfile	Loads the merchant code data file generated by SAGETREF into memory.
merchcode_lookupMerchcode_lookup	Looks up the merchant code data that we are interested in it.
partner_loadfileMerchcode_loadfile	Loads the partner data file generated by SAGETREF into memory.
partner_lookupMerchcode_lookup	Looks up the partner data that we are interested in it.
supplier_loadfileMerchcode_loadfile	Loads the supplier data file generated by SAGETREF into memory.
supplier_lookupMerchcode_lookup	*Looks up the supplier data that we are interested in it.*

IX. Input Specifications

The input files for SKU, Wastage, UPC, Variable UPC, Store Day, Promotions, Code Types, and Errors are all documented in Batch Design – SAGETREF.doc.

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

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

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

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

X. Output Specifications

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

The filename convention for storing missing transactions between invocations of SAIMPTLOG is tdup*_store_businessdate*.dat.

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

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

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

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

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

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

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

Table Name	Column Name	Field Type	Field Width	Position	Description
	Update_datetime	date	14	391:404	Format is YYYYMMDDHH24MISS
	Error_ind	char	1	405:405	
Sa_tran_item	Tran_seq_no	Integer external	20	1:20	
	Item_seq_no	Integer external	4	21:24	
	Item_status	char	6	25:30	
	Item_type	char	6	31:36	
	Sku	Integer external	8	37:44	
	Upc	char	13	45:57	
	Upc_supplement	Integer external	5	58:62	
	Voucher_no	char	16	63:78	
	Item_no	char	16	79:94	
	Dept	Integer external	4	95:98	
	Class	Integer external	4	99:102	
	Subclass	Integer external	4	103:106	
	System_ind	char	1	107:107	
	Qty	decimal external	14	108:121	Includes an optional negative sign and a decimal point.
	Unit_retail	decimal external	21	122:142	Includes a decimal point.
	Override_reason	char	6	143:148	
	Orig_unit_retail	decimal external	21	149:169	Includes a decimal point.
	Tax_ind	char	1	170:170	
	Ref_no5	char	30	171:200	
	Ref_no6	char	30	201:230	
	Ref_no7	char	30	231:260	
	Ref_no8	char	30	261:290	
	Item_swiped_ind	char	1	291:291	
	Error_ind	char	1	292:292	
	Var_upc_ind	char	1	293:293	
	Var_type	char	1	294:294	
	Waste_type	char	6	295:300	
	Pump	char	8	301:308	
	Waste_pct	decimal external	12	309:320	Includes a decimal point.
	Return_reason_code	char	6	321:326	
	Salesperson	char	10	327:336	
	Expiration_date	Date	8	337:344	Format is YYYYMMDD
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	4	35:38	
	Discount_type	char	6	39:44	
	Coupon_no	char	16	45:60	
	Coupon_ref_no	char	16	61:76	
	Qty	decimal external	14	77:90	Includes an optional negative sign and a decimal point.
	Unit_discount_amt	decimal external	21	*91:111*	Includes a decimal point.
	Ref_no13	char	30	112:141	
	Ref_no14	char	30	142:171	
	Ref_no15	char	30	172:201	
	Ref_no16	char	30	202:231	

Table Name	Column Name	Field Type	Field Width	Position	Description
	Error_ind	char	1	232:232	
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	
	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	

Table Name	Column Name	Field Type	Field Width	Position	Description
	Attrib_value	char	6	31:36	
Sa_error	Error_seq_no	Integer external	20	1:20	
	Store_day_seq_no	Integer external	20	21:40	
	Bal_group_seq_no	Integer external	20	41:60	
	Total_seq_no	Integer external	20	61:80	
	Tran_seq_no	Integer external	20	81:100	
	Error_code	char	25	101:125	
	Key_value_1	Integer external	4	126:129	
	Key_value_2	Integer external	4	130:133	
	Rec_type	char	6	134:139	
	Store_override_ind	char	1	140:140	
	Hq_override_ind	char	1	141:141	
	Update_id	char	30	142:171	
	Update_datatime	date	14	172:185	Format is YYYYMMDDHH24MISS
	Orig_value	char	50	186:235	
Sa_missing_tran	Miss_tran_seq_no	Integer external	20	1:20	
	Store_day_seq_no	Integer external	20	21:40	
	Register	char	5	41:45	
	Tran_no	Integer external	10	46:55	
	status	char	6	56:61	

XI. 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 Method

focuses

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.

XII. Scheduling Considerations

Processing Cycle: Anytime – Sales Audit 9.0 is a 24/7 system.

Scheduling Diagram: These programs (SAIMPTLOG, SQL*Loader 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

XIII. 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.

XIV. 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.

XV. 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.

XVI. 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.

XVII. Design Assumptions

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

XVIII. Outstanding Design Issues

All requirements, functional or technical issues that arise during the design of this functional area must be documented in this section. Each issue should remain on this document even if the issues has been resolved or deferred. The issue, description, status and resolution should all be maintained in this section. This section is included with the intent of acting as a worksheet that will track design and provide rationale for the decisions made during the design phase. List any outstanding issues that have been identified in this phase that need to be carried forward to the next phase(s).

Description	*Priority (High, Moderate, Low; if available)*	*Issue Log Updated?*

XIX. References

Interface File – RTLOG.doc
Interface File – SA VOUCH.doc

XX. Batch Detailed Design Walkthrough

The Batch Detailed Design document must be reviewed by Retek project representatives or alternates (if appropriate, client representatives also). Whether walkthroughs occur at one time with a single group or via parallel or sequential approvals, walkthroughs are required. Not all projects require the same level of scrutiny, but that level of scrutiny must be determined and managed from the beginning.

Retek representatives

- Project Sponsor
- Business Unit Manager
- Project Lead
- Product Manager / Strategy
- Business Analysts
- Database Analysts
- Research & Development
- Quality Control
- Documentation
- Training
- Customer Support

Only when appropriate **Client representatives**

- Business
- Technical
- End-user; those who provide / enter input, those who use outputs
- Operations
- Support

XXI. Appendix

Appendixes are included as necessary. They might include an updated glossary, derived from the Batch Detailed Design glossary, project schedules or other items interrupting the flow of this document.