Retail passion. Innovative solutions.

Retek® Merchandising System™ 10.1.10

Operations Guide Addedum

Corporate Headquarters:

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

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

Fax:
+1 612 587 5100

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

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

Fax:
+44 (0)20 7563 46 10

Printed in the United States of America.

Customer Support

Customer Support hours

Customer Support is available 7x24x365 via email, phone, and Web access.

Depending on the Support option chosen by a particular client (Standard, Plus, or Premium), the times that certain services are delivered may be restricted. Severity 1 (Critical) issues are addressed on a 7x24 basis and receive continuous attention until resolved, for all clients on active maintenance. Retek customers on active maintenance agreements may contact a global Customer Support representative in accordance with contract terms in one of the following ways.

Contact Method	Contact Information
E-mail	support@retek.com
Internet (ROCS)	rocs.retek.com Retek's secure client Web site to update and view issues
Phone	+1 612 587 5800

Toll free alternatives are also available in various regions of the world:

Australia	+1 800 555 923 (AU-Telstra) or +1 800 000 562 (AU-Optus)
France	0800 90 91 66
Hong Kong	800 96 4262
Korea	00 308 13 1342
United Kingdom	0800 917 2863
United States	+1 800 61 RETEK or 800 617 3835
Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

When contacting Customer Support, please provide:

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

Contents

Interface file - SA RTLOG

ReSA 10.0 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

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	Business Date	Char(8)	Business Date to process	Business date of transactions. (YYYYMMDD).	Y	Left/None
	Location Number	Char(10)	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

Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type.	Y	Left/Blank
	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

Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type.	Y	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 when value is present. Blank when no value is sent.

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

Transaction Customer	File Type Record Descriptor	Char(5)	TCUST	Identifies file record 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(20)		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
	Item number type	Char(6)	Refer to 'UPCT' code_type for a list of valid codes.	Identifies type of item number if item type is ITEM or REF.	N	Left/Blank
	Format ID	Char(1)	VPLU format ID.	Used to interpret VPLU items.	N	Left/Blank
	Item	Char(25)	Item identifier	Identifies merchandise item.	N	Left/Blank
	Reference Item	Char(25)	Item identifier	Identifies sub-transaction level merchandise item.	N	Left/Blank
	Non-Merchandise Item	Char(25)	Item identifier	Identifies non-merchandise item.	N	Left/Blank
	Voucher	Char(16)		Gift certificate number	N	Right/0
	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

Transaction Item	File Type Record Descriptor	Char(5)	TITEM	Identifies file record type.	Y	Left/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
	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
	SellingUnit of Measure	Char(4)		Unit of measure of item’s quantity.	Y	Left/None
	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

Transaction Item	File Type Record Descriptor	Char(5)	TITEM	Identifies file record type.	Y	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	
	Drop Ship Ind	Char(1)	Refer to 'YSNO' code type for a list of valid codes.	Indicates whether item is part of a drop shipment.	Y	Left/None

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(10)		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

Item Discount	File Type Record Descriptor	Char(5)	IDISC	Identifies file record type	Y	Left/Blank
	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

Transaction Tax	File Type Record Descriptor	Char(5)	TTAX	Identifies file record type	Y	Left/Blank
	Tax Amount	Number(20)		Amount of tax charged for this tax code type in a transaction with 4 implied decimal places.	Y	Right/0
	Reference Number 17	Char(30)		Generic reference number.	N	Left/Blank
	Reference Number 18	Char(30)		Generic reference number.	N	Left/Blank
	Reference Number 19	Char(30)		Generic reference number.	N	Left/Blank
	Reference Number 20	Char(30)		Generic reference number.	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

Transaction Tender	File Type Record Descriptor	Char(5)	TTEND	Identifies file record type	Y	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(5)		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
	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

Transaction Tender	File Type Record Descriptor	Char(5)	TTEND	Identifies file record type	Y	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 Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type	Y	Left/Blank
	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.

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.

1 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

Transaction Type	Includes item records?	Includes tender records?	Includes tax records?	Includes customer records?
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

2 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.

3 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
	Item Number Type	UPCT
	Quantity Sign	SIGN
	Taxable Indicator	YSNO
	Price Override Reason Code	ORRC
	Item Swiped Indicator	YSNO
	Return Reason Code	SARR

Record Name	Field Name	Code Type
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

4 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.

5 Store number is validated against the STORE table.

6 Numeric fields are checked for non-numeric characters.

7 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.

Treatment of vouchers

- If an item sold is a gift certificate (Transaction Item, Voucher field has a value), issued information is written to the SA_VOUCHER table.
- 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.
- 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.
- 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.

Other validations/points of interest

- A salesperson in the TITEM record takes precedence over the salesperson in the THEAD record.
- If an item sold is a sub-transaction (REF) item (Transaction Item, reference item field has a value and item does not), it will be converted to the corresponding transaction level item (ITEM).
- If an item sold is an ITEM (Transaction Item, item field has a value), it will be validated against RMS item tables.
- The corresponding Department, Class, Subclass, and Taxable Indicator will be selected from the RMS tables and populated for an item.

The balancing level determines whether the register or the cashier fields are required.

- If the balancing level is ‘R’egister, then the register field on the THEAD must be populated.
- If the balancing level is ‘C’ashier, then the cashier field on the THEAD must be populated.
- If the balancing level is ‘S’tore, then neither field is required to be populated.

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’.

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.

- 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.

Record Type	Requirements
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.

- 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

- 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.

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.

Transaction of type 'RETURN'

This transaction is generated when a customer returns an item.

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

- 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

- 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.

Transaction of type 'EEXCH'

This transaction is generated when there is an even exchange.

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

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.

No tender changes hands in this transaction.

- 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

Transaction of type 'PAIDIN'

This type of transaction has only one TTEND record.

A reason code is required.

- 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

Transaction of type 'PAIDOU'

This type of transaction has only one TTEND record.

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.

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.

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.

- 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

Transaction of type 'PULL'

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

This type of transaction has only one TTEND record.

- 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

Transaction of type 'LOAN'

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

This type of transaction has only one TTEND record.

- 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

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.

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

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

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.

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

Transaction of type 'METER'

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

This type of transaction has only TITEM records.

- 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

Reference Number Column	Meaning	Req?
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

Transaction of type 'PUMPT'

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

Transactions of type 'TANKDP'

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

This type of transaction has only TITEM records.

- 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

Transaction of type 'DCLOSE'

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

Note: 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.

Stock Upload Conversion [lifstkup]

Design Overview

This program converts the Nautilus inventory balance upload file into the Retek standard flat file for stkupld.pc to process.

This program verifies that the inventory data is for the requested cycle count and warehouse before proceeding. Other data needed for the RMS flat file will be obtained from RMS tables and inserted in the RMS flat file.

Scheduling Constraints

Processing Cycle:	N/A
Scheduling Diagram:	This program should run before stkupld.pc and after the Nautilus inv_bal_upload.sh program.
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Program Flow

N/A

Function Level Description

Init()

Obtain the cycle_count from tables stake_head and stake_location by matching the warehouse location and the stocktake_date with combination of the Nautilus input file, location(DC) and the transaction date/time (date portion only).

If no match can be made, the program will terminate.

Call restart init

Write FHEAD record.

Process()

Loop through and read each line from the Nautilus file.

Format the data for the output file.

Call write_fdetl() function.

Write_fdetl()

Call check_catch_weight() function

Write a transaction record (FDETL) to the Retek flat file for each line found in the Nautilus flat file.

Check_catch_weight()

Called within write_fdetl to check if item is check weight. Copies total weight value if it is a check weight item, otherwise copies total units.

Final ()

Write a file trailer record (FTAIL).

Close files

Call restart close

Write status to log file

I/O Specification

Both input and output file should be accepted as a runtime parameter at the command line.

Input format:

DC_DEST_ID	11 - Number(10) + 1 for trailing space.	Unique identifier for the DC
TRANSACTION_DATE	13 - Date(12) + 1 for trailing space.	Date of the run
ITEM_ID	26 - Varchar2(25) + 1 for trailing space.	Uniquely identifies the item.
AVAILABLE_QTY	15 - Number(12) + 1 for leading sign and + 1 for decimal and + 1 for trailing space.	Units available for distribution
DISTRIBUTED_QTY	14 Number(12) + 1 for decimal and + 1 for trailing space.	Units distributed include: Units distributed but not yet picked, units picked but not yet manifested, units manifested but not yet shipped.
RECEIVED_QTY	14 Number(12) + 1 for leading sign and + 1 for decimal and + 1 for trailing space.	Units received but not put away.
TOTAL_QTY	14 Number(12) + 1 for decimal and + 1 for trailing space.	Sum of all units that physically exist: container status of: I, D, M, R, T, X.
AVAILABLE_WEIGHT	15 - Number(12,4) + 1 for leading sign + 1 for decimal + 1 for trailing space.	Weight available for distribution of catch weight items.

RECEIVED_WEIGHT	14 - Number(12,4) + 1 for decimal + 1 for trailing space.	Weight received but not put away for catch weight items.
DISTRIBUTED_WEIGHT	14 - Number(12,4) + 1 for decimal + 1 for trailing space.	Weight distributed includes: weight distributed but not yet picked, weight picked but not yet manifested, weight manifested but not yet shipped (value only catch weight items)
TOTAL_WEIGHT	13 - Number(12,4) + 1 for decimal.	Sum of all weight that physically exist: container status of: I, D, M, R, T, X. For catch weight items.

Output Format:

Record Name	Field Name	Field Type	Description
File Header	file type record descriptor	Char(5)	hardcode 'FHEAD'
	file line identifier	Number(10)	Id of current line being processed., hardcode '000000001'
	file type	Char(4)	hardcode 'STKU'
	file create date	Date(14) YYYYMMDDHHMISS	date written by convert program
	stocktake_date	Date(14) YYYYMMDDHHMISS	stake_head.stocktake_date
	cycle count	Number(8)	stake_head.cycle_count
	loc_type	Char(1)	hardcode 'W'
	location	Number(10)	stake_location.wh
Transaction record	file type record descriptor	Char(5)	hardcode 'FDETL'
	file line identifier	Number(10)	Id of current line being processed, internally incremented
	item type	Char(3)	hardcode 'ITM'

Record Name	Field Name	Field Type	Description
	item value	Number(25)	item id
	inventory quantity	Number(12,4)	total units or total weight
	location description	Char(30)	NULL
File trailer	file type record descriptor	Char(5)	hardcode ‘FTAIL’
	file line identifier	Number(10)	Id of current line being processed, internally incremented
	file record count	Number(10)	Number of detail records.

Technical Issues

N/A

Like Store [likestore]

Design Overview

When a new store is created in RMS there is an option to specify a like store. When storeadd batch is run it sets the store open date and close date of all the like stores far in the future, so that those records will be picked up in the likestore batch. Likestore batch creates item location relationships for all the items in the existing store with new store. The likestore batch will process like stores and sets the store open and close dates back to original date in the post process. User can specify whether to copy the Replenishment information, delivery schedules and activity schedules from the existing store, which will be copied in the likestore post process. So it is necessary to run the storeadd, likestore and likestore post in the same order to successfully add all the stores in to RMS.

Likestore batch uses multi-threading by department along with array processing to copy item expense information. It also utilizes array processing to fetch all items associated to the likestore and their attributes. The array of these items and their attributes is then looped through, with the NEW_ITEM_LOC procedure being called for each item to create the new relationship.

Scheduling Constraints

Processing Cycle:	Ad Hoc Phase
Scheduling Diagram:	N/A
Pre-Processing:	storeadd.pc
Post-Processing:	prepost(likestore post)
Threading Scheme:	Table based processing, multithreading on Department.

Restart/Recovery

The logical unit of work is store, item, pack indicator. The following two cursors will keep track of store, item, and pack indicator in the restart book mark. The c_add_store cursor restart the program based on store and c_get_items will restart the program based on item, pack indicator.

```
EXEC SQL DECLARE c_add_store CURSOR for
       SELECT sa.store,
              sa.like_store,
              ROWIDTOCHAR(st.rowid)
         FROM store_add sa,
              store st
        WHERE sa.store = st.store
          AND st.store_open_date = sa.store_open_date + 500000
          AND st.store_close_date = sa.store_open_date + 500000
          AND (sa.store > NVL(:ps_restart_store,-999) OR
               sa.store = :ps_restart_store)
     ORDER BY sa.store;

EXEC SQL DECLARE c_get_items CURSOR FOR
```

```
SELECT il.item,
       im.item_desc,
       im.diff_1,
       im.diff_2,
       im.diff_3,
       im.diff_4,
       il.loc_type,
       il.daily_waste_pct,
       iscl.unit_cost,
       il.unit_retail,
       il.selling_unit_retail,
       il.selling_uom,
       il.status,
       il.taxable_ind,
       il.ti,
       il.hi,
       il.store_ord_mult,
       il.meas_of_each,
       il.meas_of_price,
       il.uom_of_price,
       il.primary_variant,
       il.primary_supp,
       il.primary_cntry,
       il.local_item_desc,
       il.local_short_desc,
       il.primary_cost_pack,
       il.receive_as_type,
       im.item_parent,
       im.item_grandparent,
       im.dept,
       im.class,
       im.subclass,
       im.status,
       cl.class_vat_ind,
       im.short_desc,
       im.item_level,
       im.tran_level,
```

```
            im.retail_zone_group_id,
            pzgs.zone_id,
            im.sellable_ind,
            im.orderable_ind,
            im.pack_ind,
            im.pack_type,
            im.waste_type,
            st.lang,
            il.source_method,
            il.source_wh
       FROM v_restart_dept vrd,
            store st,
            price_zone_group_store pzgs,
            item_master im,
            class cl,
            item_loc il,
            item_supp_country_loc iscl
      WHERE vrd.num_threads = TO_NUMBER(:ps_num_threads)
        AND vrd.thread_val = TO_NUMBER(:ps_thread_val)
        AND vrd.driver_value = im.dept
        AND st.store = TO_NUMBER(:is_like_store)
        AND st.store = il.loc
        AND ((im.pack_ind = NVL(:ps_restart_pack_ind, 'N') AND
im.item > NVL(:ps_restart_item, ' '))
            OR (im.pack_ind > NVL(:ps_restart_pack_ind, 'N') AND
im.item > ' '))
        AND il.item = im.item
        AND im.dept = cl.dept
        AND im.class = cl.class
        AND pzgs.store(+) = TO_NUMBER(:is_store)
    AND im.retail_zone_group_id = pzgs.zone_group_id(+)
        AND il.CLEAR_IND = 'N'
        AND il.ITEM = iscl.ITEM(+)
        AND il.LOC = iscl.LOC(+)
        AND il.primary_supp = iscl.supplier(+)
        AND il.primary_cntry = iscl.origin_country_id(+)
   ORDER BY im.pack_ind asc,
       il.item;
```

Program Flow

N/A

Function Level Description

init()

Initialize the restart variables

Get system variables (ELC indicator, VAT indicator, std_av_ind and rpm_ind)

process()

- Select values from the STORE_ADD table for stores that the storeadd.pc program has already processed, as evidenced by the store open date far in the future.
- Loop through all the likestore records and call Copy_Store_Items function for each like store record.

Copy_Store_Items()

- If the ELC indicator is “Y”, the item expenses tables are updated with the details of expenses involved in moving the items from one location to other locations. This is done using array possessing.
- C_get_items cursor will fetch all the records for the item location combination of the old store and create all the item location relationships with new store by calling the function NEW_ITEM_LOC().
- Inside the NEW_ITEM_LOC function
- Item location records are inserted for all the parent and child items, all component items in case of pack item.
- New store zone is added.
- Price history records are inserted.
- Pos mod records are inserted.
- Replenishment information, Delivery schedules and Activity schedules are copied if specified in the likestore batch post process.

Size_exp_head()

- Allocates memory to the exp_head structure

Size_exp_head_seq()

- Allocates memory to the exp_head_seq structure

Size_exp_insert()

- Allocates memory to the exp_insert structure

Size_new_itemloc()

- Allocates memory to the new_itemloc structure

free_exp_head()

- Releases the memory allocated in size_exp_head function.

free_exp_head_seq()

- Releases the memory allocated in size_exp_head_seq function.

free_exp_insert()

- Releases the memory allocated in size_exp_insert function.

free_new_itemloc()

- Releases the memory allocated in size_new_itemloc function.

final()

- This function stops restart recovery.

I/O Specification

N/A

Technical Issues

N/A

Processing Cursors

```
/* Any changes made to c_count_item_exp_head must be replicated in
c_item_exp_head */

/* The count returned in c_count_item_exp_head determines the number
of records    */

/* to be processed by c_item_exp_head. The 'FROM' and 'WHERE'
clauses must match.  */

   EXEC SQL DECLARE c_count_item_exp_head CURSOR FOR

      SELECT count(ieh.item)

        FROM v_restart_dept vrd,

             cost_zone_group czg,

             item_master im,

             item_exp_head ieh

       WHERE vrd.num_threads = TO_NUMBER(:ps_num_threads)

         AND vrd.thread_val = TO_NUMBER(:ps_thread_val)

         AND vrd.driver_value = im.dept

         AND czg.cost_level = 'L'

         AND czg.zone_group_id = im.cost_zone_group_id

         AND im.item = ieh.item

         AND (:ps_restart_item = '-999' OR :ps_restart_item is NULL)

         AND ieh.zone_group_id = czg.zone_group_id

         AND ieh.zone_id = TO_NUMBER(:is_like_store)

         AND ieh.item_exp_type = 'Z';
```

```
/* Any changes made to c_item_exp_head must be replicated in
c_count_item_exp_head */

/* The count returned in c_count_item_exp_head determines the number
of records    */

/* to be processed by c_item_exp_head. The 'FROM' and 'WHERE'
clauses must match.  */

   EXEC SQL DECLARE c_item_exp_head CURSOR FOR

      SELECT ieh.item,

             ieh.supplier,

             NVL(ieh.item_exp_seq,0),

             ROWIDTOCHAR(ieh.rowid)

        FROM v_restart_dept vrd,

             cost_zone_group czg,

             item_master im,

             item_exp_head ieh

       WHERE vrd.num_threads = TO_NUMBER(:ps_num_threads)

         AND vrd.thread_val = TO_NUMBER(:ps_thread_val)

         AND vrd.driver_value = im.dept

         AND czg.cost_level = 'L'

         AND czg.zone_group_id = im.cost_zone_group_id

         AND im.item = ieh.item

         AND (:ps_restart_item = '-999' OR :ps_restart_item is NULL)

         AND ieh.zone_group_id = czg.zone_group_id

         AND ieh.zone_id = TO_NUMBER(:is_like_store)

         AND ieh.item_exp_type = 'Z'

    ORDER BY ieh.item, ieh.supplier, ieh.item_exp_seq desc;


/*Any changes made to c_count_item_exp_head_seq must be replicated
in              */

/*c_item_exp_head_seq.  The count returned in
c_count_item_exp_head_seq determines*/

/* the number of records to be processed by c_item_exp_head_seq. The
'FROM' and     */

/* 'WHERE' clauses must match.
*/

   EXEC SQL DECLARE c_count_item_exp_head_seq CURSOR FOR

      SELECT COUNT(MAX(item_exp_seq))

        FROM item_exp_head

       WHERE item_exp_type = 'Z'
```

```
    GROUP BY item, supplier, item_exp_type;


/* Any changes made to c_item_exp_head_seq must be replicated in
*/
/* c_count_item_exp_head_seq.  The count returned in
c_count_item_exp_head_seq      */
/* determines the number of records to be processed by
c_item_exp_head_seq.         */
/* The 'FROM' and 'WHERE' clauses must match.
*/
   EXEC SQL DECLARE c_item_exp_head_seq CURSOR FOR
      SELECT item,
             supplier,
             NVL(MAX(item_exp_seq),0)
        FROM item_exp_head
       WHERE item_exp_type = 'Z'
    GROUP BY item, supplier, item_exp_type;


/* Any changes made to c_count_items must be replicated in
c_get_items        */
/* The count returned in c_count_items determines the number of
records       */
/* to be processed by c_get_items. The 'FROM' and 'WHERE' clauses
must match. */
   EXEC SQL DECLARE c_count_items CURSOR FOR
      SELECT count(im.item)
        FROM v_restart_dept vrd,
             store st,
             price_zone_group_store pzgs,
             item_master im,
             class cl,
             item_loc il,
             item_supp_country_loc iscl
       WHERE vrd.num_threads = TO_NUMBER(:ps_num_threads)
         AND vrd.thread_val = TO_NUMBER(:ps_thread_val)
         AND vrd.driver_value = im.dept
         AND st.store = TO_NUMBER(:is_like_store)
         AND st.store = il.loc
         AND ((im.pack_ind = NVL(:ps_restart_pack_ind, 'N') AND
im.item > NVL(:ps_restart_item, ' '))
```

```
              OR (im.pack_ind > NVL(:ps_restart_pack_ind, 'N') AND
im.item > ' '))
          AND il.item = im.item
          AND im.dept = cl.dept
          AND im.class = cl.class
          AND pzgs.store(+) = TO_NUMBER(:is_store)
      AND im.retail_zone_group_id = pzgs.zone_group_id(+)
          AND il.CLEAR_IND = 'N'
          AND il.ITEM = iscl.ITEM(+)
          AND il.LOC = iscl.LOC(+)
          AND il.primary_supp = iscl.supplier(+)
          AND il.primary_cntry = iscl.origin_country_id(+);
```

Clearance Pricing POS Download [pccdnld]

Design Overview

The purpose of the Clearance Download (pccdnld) module is to send clearance markdown retail prices to the point of sales system. Clearance markdowns that are to take effect within the predetermined number of days are written out to the POS_MODS table, which will be used as an interface point with the point of sale. Also, the clearance detail records are updated with the current date as a downloaded date. The number of days prior to taking effect that clearance markdowns are downloaded is maintained as a system option.

Table	Index	Select	Insert	Update	Delete
PERIOD	No	Yes	No	No	No
UNIT_OPTIONS	No	Yes	No	No	No
CLEAR_SUSP_DETAIL	No	Yes	No	Yes	No
CLEAR_SUSP_HEAD	No	Yes	No	No	No
PRICE_ZONE_GROUP_STORE	No	Yes	No	No	No
ITEM_LOC	No	Yes	No	No	No
ITEM_MASTER	No	Yes	No	No	No
ITEM_ZONE_PRICE	No	Yes	No	No	No
POS_MODS	No	No	Yes	No	No
RESTART_CONTROL	No	Yes	No	No	No

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Phase 1

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Clearance

Restart Recovery

Logical Unit of Work (recommended Commit checkpoints)

Driving Cursor

```
EXEC SQL DECLARE c_clear CURSOR FOR
    SELECT csd.item,
                  csd.clearance,
                  TO_CHAR(csd.active_date, 'DDMMYYYY'),
                  csd.zone_group_id,
                  csd.zone_id,
                  csd.unit_retail,
                  pzgs.store,
                  ROWIDTOCHAR(csd.rowed),
                  csd.selling_uom,

     FROM clear_susp_detail csd,
                clear_susp_head csh,
                price_zone_group_store pzgs,
                item_master im,
                restart_control rc
  WHERE :prc_ext_days            >= (csd.active_date -
TO_DATE(:vdate, 'DDMMYYYY'))
       AND csd.active_date           >= TO_DATE(:vdate + 1,
'DDMMYYYY')
       AND csh.status                      = 'A'
       AND csd.downloaded_date is NULL
       AND csd.clearance                = csh.clearance
       AND pzgs.zone_group_id      = csd.zone_group_id
       AND pzgs.zone_id                = csd.zone_id
       AND csd.item                        = im.item
       AND rc.driver_name             = :ora_restart_driver_name
       AND rc.num_threads            = :ora_restart_num_threads
       AND rc.program_name         = ‘pccdnld’
       AND restart_thread_return(csh.clearance, rc.num_threads) =
:ora_restart_thread_val
       AND im.status                       = ‘A’
       AND im.item_level               = im.tran_level
       AND im.pack_ind                 = ’N’
  ORDER BY csd.clearance,
```

```
csd.item,
csd.zone_group_id,
csd.zone_group_id;
```

Program Flow

N/A

Shared Modules

UOM_SQL.CONVERTS

POS_UPDATE_SQL.POS_MODS_INSERT

Function Level Description

init()

This function will, in addition to performing standard restart tasks, fetch the pos_extract_days from the unit_options table, current date from the period table and the current date plus 1 also from the period table. Size_arrays function is also called to allocate memory size to arrays.

process()

This function selects the impending clearance records for processing. The records are selected into arrays (size determined by commit_max_ctr on the RESTART_CONTROL table) and then each record in the array is processed by a for-loop.

Inside this internal loop, the ITEM information is retrieved from the ITEM_MASTER table (item, status, & pack_ind). If the status is approved, then the set of clearance records are considered for processing. If the clearance number has changed since the last check then the update_clear_susp_detail() function is called to update the header status. When the CLEAR_SUSP_DETAIL rowid changes then a row is added to an array that will update the status of the CLEAR_SUSP_DETAIL table.

After the internal loop has processed the individual records, the insert_tran_type() function is called to insert POS_MODS records and the clear_susp_detail rows that have completed processing are updated by calling the update_clear_susp_detail() function.

convert_old_unit_retail()

Calls the UOM_SQL.CONVERTS to evaluate whether the classes are the same or different, and then it will call the corresponding internal function to convert between the given unit of measures.

insert_tran_type()

Calls the POS_UPDATE_SQL.POS_MODS_INSERT function to insert the clearance records into pos_mods. If the is_clear_ind column in the ITEM_LOC table is not NULL, the tran_type is 17, otherwise, the tran_type is 16. The package first validates input parameters, making sure fields are null or not null as required. It then does an insert into the table with appropriate values as determined by the tran_type.

get_pc_unit_retail()

The sole function of this function will simply be to populate the arrays for current unit retail and price change from the PRICE_SUSP_DETAIL table, with references to the CLEAR_SUSP_DETAIL and PRICE_SUSP_DETAIL tables.

update_clear_susp_detail()

The CLEAR_SUSP_DETAIL table is updated, the downloaded_date is set to the current date, for all of the records whose rowids are in the update array.

size_arrays()

This function allocates memory sizes to the arrays that have been fetched into by the driving cursor.

final()

This function closes restart recovery.

I/O Specification

N/A

Technical Issues

N/A

Clearance Reset Pricing POS Extract [pccrdnld]

Design Overview

The Clearance Reset Pricing Information (pccrdnld) module is used to send clearance reset pricing information to the point of sale system. When a clearance event is within a pre-determined number of days of its reset date, this program will gather current clearance pricing information and the pricing information to which an item will be reset. This information will be written out to a table that will be used as an interface point with the point of sale. Clearance records will be updated to indicate that they have had reset prices downloaded to the point of sale by setting a reset downloaded date with the current date.

Tables	Index	Select	Insert	Update	Delete
PERIOD	No	Yes	No	No	No
UNIT_OPTIONS	No	Yes	No	No	No
CLEAR_RESET_CALC	No	Yes	No	Yes	No
CLEAR_SUSP_HEAD	No	Yes	No	No	No
CLEAR_SUSP_DETAIL	No	Yes	No	No	No
PRICE_ZONE_GROUP_STORE	Yes	Yes	No	No	No
POS_MODS	No	No	Yes	No	No
ITEM_MASTER	Yes	Yes	No	No	No
ITEM_LOC	Yes	Yes	No	No	No
ITEM_ZONE_PRICE	Yes	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Phase 1

Scheduling Diagram: N/A

Pre-Processing: pcdnld should complete processing before this module begins

Post-Processing: N/A

Threading Scheme: Clearance

Restart Recovery

The driving cursor was changed because of changes to the tables and also the system indicator is no longer used.

```
EXEC SQL DECLARE c_get_reset CURSOR FOR
           SELECT crc.clearance,
              crc.item,
              TO_CHAR(crc.reset_date, 'DDMMYYYY'),
              pzgs.store,
              ROWIDTOCHAR(crc.rowid),
              im.item_level,
              ';' || TO_CHAR(crc.clearance) ||
              ';' || crc.item ||
              ';' || TO_CHAR(crc.zone_group_id) ||
              ';' || TO_CHAR(crc.zone_id)
         FROM clear_reset_calc crc,
              clear_susp_head csh,
              price_zone_group_store pzgs,
              item_master im,
              v_restart_clearance rv
        WHERE crc.reset_date is NOT NULL
          AND (
                 (csh.status = 'C'
                  AND EXISTS (SELECT 'x'
                              FROM clear_susp_detail csd
                             WHERE csd.clearance     = csh.clearance
                               AND csd.item          = crc.item
                               AND csd.zone_group_id =
crc.zone_group_id
                               AND csd.zone_id       = crc.zone_id
                               AND csd.downloaded_date is NOT NULL))
               OR
                  (csh.status        = 'A'
                  AND :pd_pos_extract_days >= (crc.reset_date -
to_date(:ps_vdate,'DDMMYYYY'))              )
               )
          AND csh.clearance     = crc.clearance
          AND crc.zone_group_id = pzgs.zone_group_id
          AND crc.zone_id       = pzgs.zone_id
```

```
       AND crc.reset_downloaded_date is NULL
       AND crc.item          = im.item
       AND im.status         = 'A'
       AND im.pack_ind       = 'N'
       AND im.item_level     = im.tran_level
       AND rv.driver_value   = csh.clearance
       AND rv.driver_name    = :ps_restart_driver_name
       AND rv.num_threads    = :pi_restart_num_threads
       AND rv.thread_val     = :pi_restart_thread_val
ORDER BY crc.clearance,
         crc.item,
         crc.zone_group_id,
         crc.zone_id;
```

Program Flow

N/A

Shared Modules

POS_UPDATE_SQL.POS_MODS_INSERT

Function Level Description

Init()

Initialize restart recovery.

Determine pos_extract_days, vdate and vdate + 1.

Process()

Select all clearance details with status in 'A' or 'C' to send clearance reset pricing information to the point of sale system. It will call function get_item_info to obtain the dept, selling_unit_retail, selling_uom, multi_selling_uom, multi_unit_retail, zone_group_id and zone_id. If the item/store location exists, it will call function price_check, pos_mods and update_clear_reset_calc functions. Price_check is a function to check for any extracted price_susp records that exist. Pos_mods is a function which calls POS_UPDATE_SQL.POS_MODS_INSERT package. It writes price resets to the POS_MODS table. A reset price is one that existed prior to a clearance price period. The reset prices are scheduled to take effect in a pre-determined number of days. Update_clear_reset_calc is a function which updates the reset_downloaded_date of clear_reset_calc table with the current vdate.

Get_Item_Info()

This function gets the dept, selling_unit_retail, selling_uom, multi_selling_uom, multi_unit_retail, zone_group_id and zone_id from the ITEM_MASTER, ITEM_ZONE_PRICE, PRICE_ZONE_GROUP_STORE and ITEM_LOC tables.

Price_Check ()

This function selects item_parent, item_grandparent, diff_1, diff_2, diff_3, diff_4 from the ITEM_MASTER table. Use the retrieved values in selecting the unit_retail, multi_units, multi_unit_retail, selling_uom and multi_selling_uom from PRICE_SUSP_HEAD and PRICE_SUSP_DETAIL tables.

Pos_Mods ()

This function calls POS_UPDATE_SQL.POS_MODS_INSERT package. It writes price resets to the POS_MODS table. It uses 18 as tran_type to remove clearance item and reset.

Update_Clear_Reset_Calc()

This function will update the reset_downloaded_date of clear_reset_calc table with the current vdate when the rowid changes, this is the equivalent of the restart_string changing and will not affect restart_recovery

Final()

This function closes restart recovery.

I/O Specification

N/A

Technical Issues

N/A

Retek Clearance Reset Price Update [pccrext]

Design Overview

The function of this program is to reset the SKU/store retail prices back to the regular retail prices. Clearance events with reset dates on the following business day are selected and the appropriate sku/location tables are updated with the regular retail price and a new clearance indicator value. Transaction-level stock ledger, price history, supplier history and clearance sell through records are also written.

Tables	Index	Select	Insert	Update	Delete
PERIOD	No	Yes	No	No	No
CLEAR_SUSP_DETAIL	No	Yes	No	No	No
CLEAR_SUSP_HEAD	No	Yes	No	Yes	No
WIN_SKUS	Yes	Yes	No	No	No
WIN_WH	Yes	No	No	Yes	No
WIN_STORE	Yes	Yes	No	Yes	No
RAG_STYLE	Yes	Yes	No	No	No
RAG_SKUS	Yes	Yes	No	No	No
RAG_SKUS_WH	Yes	No	No	Yes	No
RAG_SKUS_ST	Yes	Yes	No	Yes	No
TRAN_DATA	No	No	Yes	No	No
SUP_DATA	No	No	Yes	No	No
PRICE_HIST	No	No	Yes	No	No
ITEM_ZONE_PRICE	Yes	Yes	No	No	No
PRICE_ZONE_GROUP_STORE	Yes	Yes	No	No	No
CLEAR_RESET_CALC	No	Yes	No	No	No
SKU_SUPPLIER	Yes	Yes	No	No	No

Indexes: WIN_SKUS(sku),
WIN_WH(sku),
WIN_STORE(sku), (sku, store),
RAG_STYLE(style),
RAG_SKUS(style),
RAG_SKUS_WH(style),
RAG_SKUS_ST(style, store),
ITEM_ZONE_PRICE(zone_group_id, zone_id),
PRICE_ZONE_GROUP_STORE(zone_group_id, zone_id),
(zone_group_id, store),
SKU_SUPPLIER(sku)

Scheduling Constraints

Processing Cycle:	Phase 3
Scheduling Diagram:	N/A
Pre-Processing:	pcext and pccext must complete before this module begins
Post-Processing:	N/A
Threading Scheme:	Clearance

Restart Recovery

```
EXEC SQL DECLARE c_reset CURSOR FOR
    SELECT csh.clearance,
           csh.status,
           csh.reason,
           crc.sku,
           crc.zone_group_id,
           crc.zone_id,
           TO_CHAR(crc.reset_date, 'DDMMYYYY'),
           pzgs.store,
           ';'||TO_CHAR(csh.clearance)||
           ';'||TO_CHAR(crc.sku)||
           ';'||TO_CHAR(pzgs.store)
FROM v_restart_clearance rv,
           price_zone_group_store pzgs,
           clear_reset_calc crc,
           clear_susp_head csh
WHERE ((csh.status = 'A'
      AND  crc.reset_date = to_date(:tomorrow,   'DDMMYYYY'))
         OR (csh.status = 'C'
      AND EXISTS (SELECT 'x'
                                  FROM clear_susp_detail csd
```

```
                                  WHERE csd.clearance =
csh.clearance
               AND csd.sku = crc.sku
                                        AND csd.zone_group_id =
crc.zone_group_id
                                        AND csd.zone_id = crc.zone_id
                                        AND csd.downloaded_date is not
null)))
       AND csh.clearance     = crc.clearance
       AND crc.zone_group_id = pzgs.zone_group_id
       AND crc.zone_id     = pzgs.zone_id
       AND rv.driver_value = csh.clearance
       AND rv.driver_name  = :ora_restart_driver_name
       AND rv.num_threads  = :ora_restart_num_threads
       AND rv.thread_val   = :ora_restart_thread_val
       AND (csh.clearance > NVL(:ora_restart_clearance, -999)
                 OR  (csh.clearance = :ora_restart_clearance
       AND (crc.sku > :ora_restart_sku OR  (crc.sku =
:ora_restart_sku
                 AND (pzgs.store > :ora_restart_store)))))
  ORDER BY csh.clearance, sku, store;
```

Program Flow

main()

{

accept the userid/password as an argument from the command line

init()

initialize restart variables

copy start string variables from the restart_start_array

fetch today’s date and currency code

process()

declare driving cursor

Initialize temporary variables

While(1)

{

Ensure that item is not a pack item

Get item information

Insert records into tran_data, sup_data, price_hist tables

update the item_loc table by resetting the unit retail to the original unit retail and set clear_ind = ‘N’

}

for the last clearance event check if all items have been reset if so, update clear_susp_head.status to completed ('O')

final()

close restart recovery

}

Shared Modules

- **ITEM_ATTRIB_SQL.GET_STANDARD_UOM** – This packaged function will return the standard UOM and the UOM class of the item passed.
- **UOM_SQL.CONVERT** – This packaged function will convert the unit retail between the given unit of measures.
- **PRCCHGTYP** – This stored procedure is used to determine the tran_type
- **STKLEDGR_SQL.TRAN_DATA_INSERT** – This packaged function performs the insertion of records into tran_data
- **CURRENCY_SQL.CONVERT** – This packaged function will convert the dollar amount to primary currency

Function Level Description

init()

Initialize restart recovery.

Determine the vdate and the currency code.

process()

Select all clearance events with reset date of tomorrow. It will call the function check_item to ensure that the item is not a pack item. Then it will call the function get_item_info to retrieve clearance items dept, class, subclass, and primary item supplier. Update_item_loc is then called to update the item_loc table to reset the unit_retail and set the clear_ind = ‘Y’, insert into tran_data, sup_data and price_hist tables. Lasty, the function update_clear_head function is called check if all the items have been reset. If so, clear_susp_head.status is updated to completed (‘O’).

check_item()

This function checks the item to ensure that it is not a pack item since pack items are not allowed for clearance.

get_item_info()

This function selects the items dept, class, subclass, and supplier and fetches them into variables to be inserted into tables later in program.

update_item_loc()

This function selects the unit retail, in transit qty, unit cost and stock on hand from the ITEM_LOC and ITEM_LOC_SOH tables. It then converts the new reset unit retail to the standard unit retail by calling the get_reset_retail function. The function also calls calc_tran_type to compute tran_type, insert_tran to insert into the TRAN_DATA and SUP_DATA tables, and price_hist to insert into the PRICE_HIST table. The function also updates the ITEM_LOC table with the new unit retail, reset selling uom, reset selling unit retail and set the clear_ind = 'N'.

get_reset_retail()

This function will retrieve the reset retail price from the ITEM_ZONE_PRICE table and it will also check if a price change has been extracted. If a price change has been extracted, then this would be the new unit retail price. It would then call the ITEM_ATTRIB_SQL.GET_STANDARD_UOM package to retrieve the standard uom and UOM class of the item. The new selling unit retail will then be converted to the standard unit retail by calling the UOM_SQL.CONVERT package.

calc_tran_type()

This function computes the tran type by calling the stored procedure PRCCHGTYP.

insert_tran()

If the selected item's stock qty > 0, this function calls the package STKLEDGR_SQL.TRAN_DATA_INSERT to insert into the TRAN_DATA table. If the local currency is not the same as the primary currency, this function calls function convert_currency to convert to the primary currency. Then insert into the SUP_DATA table. If there is a second tran type and tran type amt, then the function calls STKLEDGR_SQL.TRAN_DATA_INSERT again and inserts into the SUP_DATA table again.

convert_currency()

Function is called to compute the dollar amount in primary currency by calling the package CURRENCY_SQL.CONVERT.

price_hist()

This function inserts into the PRICE_HIST table.

update_clear_head()

This function will update the clear_susp_head table where all clearance details has been reset or with a previous status of 'C'. Status of the clearance will be set to 'O'

final()

This function closes restart recovery.

I/O Specification

N/A

Technical Issues

N/A

PO Subscription API

Functional Area

PO subscription

Design Overview

RMS will expose an API that will allow external systems to create, edit, and delete purchase orders within RMS. The transaction will be performed immediately upon message receipt so success or failure can be communicated to the calling application.

Purchase order messages will be sent across the Retek Integration Bus (RIB). POs can be created, modified or deleted at the header or the detail level, each with its own message type.

Consume Module

Filename: rmssub_xorders/b.pls

```
RMSSUB_XORDER.CONSUME
(O_status_code      IN OUT    VARCHAR2,
O_error_message     IN OUT    RTK_ERRORS.RTK_TEXT%TYPE,
I_message           IN        RIB_OBJECT,
I_message_type      IN        VARCHAR2)
```

This procedure will need to initially ensure that the passed in message type is a valid type for purchase order messages. The valid message types for purchase order messages are listed in a section below.

If the message type is invalid, a status of "E" should be returned to the external system along with an appropriate error message informing the external system that the status is invalid.

If the message type is valid, the generic RIB_OBJECT needs to be downcast to the actual object using the Oracle's treat function. There will be an object type that corresponds with each message type. If the downcast fails, a status of "E" is returned to the external system along with an appropriate error message informing the external system that the object passed in is invalid.

If the downcast is successful, then consume needs to verify that the message passes all of RMS's business validation. It calls the RMSSUB_XORDER_VALIDATE.CHECK_MESSAGE function to determine whether the message is valid. If the message passed RMS business validation, then the function returns true, otherwise it returns false. If the message fails RMS business validation, a status of "E" is returned to the external system along with the error message returned from the CHECK_MESSAGE function.

Once the message has passed RMS business validation, it is persisted to the RMS database. It calls the RMSSUB_XORDER_SQL.PERSIST() function. If the database persistence fails, the function returns false. A status of "E" is returned to the external system along with the error message returned from the PERSIST() function.

Once the message has been successfully persisted, there is nothing more for the consume procedure to do. A success status, "S", is returned to the external system indicating that the message has been successfully received and persisted to the RMS database.

Business Validation Module

Filename: rmssub_xordervals/b.pls

It should be noted that some of the business validation is referential or involves uniqueness. This validation is handled automatically by the referential integrity constraints and the unique indexes implemented on the database.

```
RMSSUB_XORDER_VALIDATE.CHECK_MESSAGE
(O_error_message      IN OUT   RTK_ERRORS.RTK_TEXT%TYPE,
O_order_rec           OUT      NOCOPY   ORDER_SQL.ORDER_REC,
I_message             IN       RIB_XORDERDESC_REC,
I_message_type        IN       VARCHAR2)
```

This overloaded function performs all business validation associated with create/modify messages and builds the order API record with default values for persistence in the order related tables. Any invalid records passed at any time results in message failure.

Like other APIs, the purchase order API expects a snapshot of the record on both a header modify and a detail modify message, instead of only the fields that are changed. For a detail create or a detail modify message, only the order number will be validated at the header level; all other header fields are ignored.

Defaulted fields that are not included in the message structure of the object must be populated in a package business record, ORDER_SQL.ORDER_REC. This record is used as input to the database DML functions in the persist package.

ORDER CREATE

- Check required fields on both header and detail nodes.
- Verify order number does NOT already exist.
- Verify attributes in the message header are correct.
- Verify attributes in the message detail are correct.
- Verify that item/supplier and item/supp/country exist for a non-pack item.
- Verify that item/supplier and item/supp/country exist for all components of a pack item.
- Create item/supplier and item/supp/country if they don’t exist for a pack item.
- Create item/supp/country/loc if it does not exist for an item/location.
- Create item/loc relation if not already exist, including creating item_loc_soh, item_supp_country_loc, and price_hist records. If a pack item is involved, these records will be created for all component items.
- Populate record ORDER_REC with message data for both header and detail.
- Default the following fields if they are populated in the message:
  - orig_ind
  - edi_po_ind
  - premark_ind
  - origin_country_id

ORDER MODIFY

- Check required fields on the header node.
- Verify order number already exists.
- Verify attributes in the message header are correct.
- Verify attributes that cannot be modified are not changed.
- Update ordloc appropriately if closing or reinstating an order.
- Populate record ORDER_REC.ORDHEAD_ROW with message data.

ORDER DETAIL CREATE

- Check required fields on the detail node.
- Verify order number already exists.
- Verify order/item/loc does NOT already exist.
- Verify that item/supplier and item/supp/country exist for a non-pack item.
- Verify that item/supplier and item/supp/country exist for all components of a pack item.
- Create item/supplier and item/supp/country if they don't exist for a pack item.
- Create item/supp/country/loc if it does not exist for an item/location.
- Create item/loc relation if not already exists, including creating item_loc_soh, item_supp_country_loc, and price_hist records. If a pack item is involved, these records will be created for all component items.
- Populate record ORDER_REC.ORDLOCS and optionally, ORDER_REC.ORDSKUS with message data.

ORDER DETAIL MODIFY

- Check required fields on the detail node.
- Verify order/item/loc already exists.
- Verify attributes that cannot be modified are not changed.
- If order quantity is reduced, verify the new order quantity is not below what has already been received plus what is being shipped or expected.
- If the order line is cancelled or reinstated via the indicators, calculate the new quantity buckets.
- Populate record ORDER_REC.ORDLOCS and optionally, ORDER_REC.ORDSKUS with message data.

RMSSUB_XORDER_VALIDATE.CHECK_MESSAGE

(O_error_message	IN OUT	RTK_ERRORS.RTK_TEXT%TYPE,
O_order_rec	OUT NO	COPY ORDER_SQL.ORDER_REC,
I_message	IN	RIB_XORDERREF_REC,
I_message_type	IN	VARCHAR2)

This overloaded function performs all business validation associated with delete messages and builds the order API record with default values for persistence in the order related tables. Any invalid records passed at any time results in message failure.

ORDER DELETE

- Check required fields.
- Verify order number already exists.
- Verify that order is not already shipped or received.
- Delete any allocations tied to the order
- Populate record ORDER_REC.ORDHEAD_ROW with the order number for delete.

ORDER DETAIL DELETE

- Check required fields.
- Verify order/item/loc already exists.
- Verify that order line is not already shipped or received.
- Delete any allocations tied to the order line.
- Populate record ORDER_REC.ORDLOCS with the order/item/location for delete.

Bulk or single DML module

Filename: rmssub_xorders/b.pls

All insert, update and delete SQL statements are located in package ORDER_SQL. The private functions call these packages.

RMSSUB_XORDER_SQL.PERSIST

(O_error_message	IN OUT	RTK_ERRORS.RTK_TEXT%TYPE,
I_order_rec	IN	ORDER_SQL.ORDER_REC,
I_message_type	IN	VARCHAR2)

This function checks the message type to route the object to the appropriate internal functions that perform DML insert, update and delete processes.

ORDER CREATE

- Inserts records in the ORDHEAD, ORDSKU, ORDLOC tables

ORDER MODIFY

- Updates a record in the ORDHEAD table.

ORDER DELETE

- Delete an order from ORDHEAD, ORDSKU, ORDLOC tables.

ORDER DETAIL CREATE

- Inserts records in the ORDLOC and optionally, ORDSKU tables

ORDER DETAIL MODIFY

- Updates records in the ORDLOC and/or ORDSKU table.
- Also verify it doesn't end up with an Approved order with 0 total order quantity.

ORDER DETAIL DELETE

- Delete records from ORDLOC and optionally, ORDSKU tables.
- Also verify it doesn't end up with an Approved order with no detail or with 0 total order quantity.

After all inserts, updates and deletes have taken place, this function will call order inventory management, order expense, deals, order rounding, OTB and bracket costing functionality for the order that was created, modified or deleted.

Message DTD

Here are the filenames that correspond with each message type. Please consult the mapping documents for each message type in order to get a detailed picture of the composition of each message.

Message Types	Message Type Description	Document Type Definition (DTD)
XorderCre	Order Create Message	XOrderDesc.dtd
XorderMod	Order Modify Message	XOrderDesc.dtd
XorderDel	Order Delete Message	XOrderRef.dtd
XorderDtlCre	Order Detail Create Message	XOrderDesc.dtd
XorderDtlMod	Order Detail Modify Message	XOrderDesc.dtd
XorderDtlDel	Order Detail Delete Message	XOrderRef.dtd

Design Assumptions

Required fields are shown in mapping document.

Many ordering functionalities that are available on-line are not supported via this API. See the SAE and the story document for a list of that. Triggers related to these functionalities should be turned off.

Tables

TABLE	SELECT	INSERT	UPDATE	DELETE
ORDHEAD	Yes	Yes	Yes	Yes
ORDSKU	Yes	Yes	Yes	Yes
ORDLOC	Yes	Yes	Yes	Yes

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_SUPPLIER	Yes	Yes	No	No
ITEM_SUPP_COUNTRY	Yes	Yes	No	No
ITEM_SUPP_COUNTRY_LOC	Yes	Yes	No	No
ITEM_LOC	Yes	Yes	No	No
ITEM_LOC_SOH	Yes	Yes	No	No
PRICE_HIST	No	Yes	No	No
ITEM_ZONE_PRICE	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
PACKITEM_BREAKOUT	Yes	No	No	No
SHIPMENT	Yes	No	No	No
SHIPSKU	Yes	No	No	No
APPT_DETAIL	Yes	No	No	No
ALLOC_HEADER	Yes	No	No	Yes
ALLOC_DETAIL	Yes	No	No	Yes
STORE	Yes	No	No	No
WAREHOUSE	Yes	No	No	No
SUPS	Yes	No	No	No
DEPS	Yes	No	No	No
CURRENCIES	Yes	No	No	No
CURRENCY_RATES	Yes	No	No	No
TERMS	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No
UNIT_OPTIONS	Yes	No	No	No
ADDR	Yes	No	No	No
OTB	No	Yes	Yes	Yes
ORD_INV_MGMT	No	Yes	Yes	Yes
ORDLOC_EXP	No	Yes	Yes	Yes
DEAL_CALC_QUEUE	No	Yes	No	Yes

reclsdly Master Batch Design

Functional Area

Reclassification

Module Affected

Reclsdly.pc

Design Overview

The Item Reclassification batch program is executed in order to reclassify items from one department, class or subclass to another. This reclassification of items into different merchandise hierarchy level is initiated or requested online in the item reclassification dialog, with an effective date specified. This program reads in the reclassification requests that are effective the following day, and for each item being reclassified, the following functions are executed:

- Checks if the item is forecastable and if it is, then checks for the existence of a domain. If the item is forecastable and no domain association to the new merchandise hierarchy level exists, reject the item (i.e. the item will not be reclassified).
- Updates the appropriate item table, e.g. item_master, with the new merchandise hierarchy.
- If an item is reclassified, the product securities of the item are then updated.

Stored Procedures / Shared Modules (Maintainability)

FORECASTS_SQL.GET_SYSTEM_FORECAST_IND:
Stored PL/SQL procedure for returning the value of the forecast_ind from the system_options table.

FORECASTS_SQL.GET_DOMAIN:
Stored PL/SQL procedure for retrieving the domain for a merchandise hierarchy.

RECLASS_SQL.ITEM_PROCESS:
Stored PL/SQL procedure for updating or inserting records into the item_master, pos_modS, and tran_data tables. If the item cannot be reclassified, the IO_RECLASS_FAILED variable returns a value of TRUE. Otherwise, it returns a value of FALSE. If reclassification failed, the function returns the reason for failure.

LOC_PROD_SECURITY_SQL.INSERT_USR_SEC:
Stored PL/SQL procedure for creating new records on to the sec_user_prod_matrix table for all users that have security access to the new item.

ITEMLIST_MC_REJECTS_SQL.INSERT_REJECTS:
Stored PL/SQL procedure for inserting reclass failed reasons into the mc_rejections table.

Input Specifications
Select from: V_RESTART_RECLASS, RECLASS_ITEM, RECLASS_HEAD, ITEM_MASTER, PACKITEM, ORDSKU, DEAL_CALC_QUEUE, DEAL_ORDER_TEMP, SYSTEM_OPTIONS, and PERIOD

Output Specifications

'Table-To-Table'

Delete from: RECLASS_HEAD, RECLASS_ITEM, SEC_GROUP_PROD_MATRIX, and SEC_USER_PROD_MATRIX

Insert into: DEAL_ORDER_TEMP

Function Level Description

init()

- This function initializes restart/recovery and fetches global options and variables. Calls size_arrays function.

process()

- This is the main control function of the program. Each reclass_head record fetched from the driving cursor is checked for domain association to the new merchandise hierarchy if the forecast_ind is 'Y'. This existence check will be used during the call to the RECLASS_SQL.ITEM_PROCESS function to determine if the record will be rejected from reclassification. Calls the process_item function to perform the item reclassification. If reclassification failed, the insert_reject_record and delete_reclass_item functions are called. Otherwise, all order numbers associated to the item are inserted into the DEAL_ORDER_TEMP table by calling update_deal_calc_queue function for later processing during prepost. Also, reclassified items are deleted from the RECLASS_ITEM table and the product securities of the items are updated by calling the process_security function. After processing, the reclass_head record is then deleted from the RECLASS_HEAD table.

delete_reclass_head()

- This function deletes the reclassification record from the RECLASS_HEAD table.

delete_reclass_item()

- This function deletes the record from the RECLASS_ITEM table based on the row_id.

check_domain_exists()

- If forecast_ind is 'Y', this function checks if a domain association to the new merchandise hierarchy exist for the given dept/class/subclass.

process_item()

- This function calls the RECLASS_SQL.ITEM_PROCESS function to perform the item reclassification.

size_arrays()

- This function sizes the fetched array to the commit size.

process_security()

- This function deletes the records of the reclassified item in the SEC_GROUP_PROD_MATRIX and SEC_USER_PROD_MATRIX tables. Calls the LOC_PROD_SECURITY_SQL.INSERT_USR_SEC function.

get_order_numbers()

- This function finds all the order numbers that are associated to the input item in the ORDSKU table.

update_deal_calc_queue()

- This function is passed an array of order numbers and a long telling the number of order numbers in the array. This function then inserts the order numbers into the DEAL_ORDER_TEMP table, prepost post processing will select from this table and insert into DEAL_CALC_QUEUE table, along with 'N' for the recalc_all_ind and override_manual_ind columns.

order_exists()

- This function checks to see whether or not the passed order number already exists in the DEAL_CALC_QUEUE or the DEAL_ORDER_TEMP table. A zero is returned if the order number is not in the DEAL_CALC_QUEUE OR in the DEAL_ORDER_TEMP table, a one if it is in the table, or a negative one if there was a SQL_ERROR found.

insert_reject_record()

- This function calls the ITEMLIST_MC_REJECTS_SQL.INSERT_REJECTS function to insert the reclassification failed reason in the MC_REJECTIONS table.

final()

- Standard Retek final function. Calls retek_close().

Scheduling Considerations

Processing Cycle:	Daily
Scheduling Diagram:	This module is scheduled to run daily during Phase 4 of the batch schedule. The requirement is for all the batch modules that update inventory and retail / cost prices to be completed before this module can run.
Pre-Processing:	Prepost (reclsdly pre)
Post-Processing:	Prepost (reclsdly post)
Threading Scheme:	If desired, this program can be threaded by reclass_no to process several reclassifications at once.

Restart/Recovery

The logical unit of work for the reclassification is reclass_no/item. The restart commit counter will need to be carefully determined by each client according to the number of item/store combinations that will be affected by the reclassification since processing is done inside the packages rather than directly from the batch program. Large reclassifications with thousands of items held at many stores need smaller commit counters to avoid reprocessing large amounts of data in the event of program failure. Small reclassifications affecting just a few item/store combinations can have much larger commit counters since fewer rows will be inserted into the database each time an item is processed.

Driving cursor:

```
         SELECT ROWIDTOCHAR(ri.rowid),
                        ri.reclass_no,
                        ri.item,       /* This is a level 1 item */
                        im.item,
                        NVL(im.item_parent,' '),
                        NVL(im.item_grandparent,' '),
                        im.item_level,
                        im.tran_level,
                        rh.to_dept,
                        rh.to_class,
                        rh.to_subclass,
                        im.dept,
                        im.class,
                        im.subclass
            FROM v_restart_reclass rv,
                        reclass_item ri,
                        reclass_head rh,
                        item_master im
          WHERE rh.reclass_no   = ri.reclass_no
                AND rh.reclass_date <= TO_DATE(:ps_vdate, 'YYYYMMDD')
                AND (ri.item = im.item
                   OR ri.item = im.item_parent
                   OR ri.item = im.item_grandparent)
                AND rv.driver_value = rh.reclass_no
                AND rv.driver_name  = :ps_driver_name
                AND rv.num_threads  = TO_NUMBER(:ps_num_threads)
                AND rv.thread_val   = TO_NUMBER(:ps_thread_val)
                AND (rh.reclass_no > NVL(:ps_restart_reclass_no, -999)
   OR
                         (rh.reclass_no = :ps_restart_reclass_no AND
                         ri.item > :ps_restart_item))
          UNION
          -- This is for simple pack
         SELECT ROWIDTOCHAR(ri.rowid),
                        ri.reclass_no,
                        ri.item,
                        im.item,
```

```
                        NVL(im.item_parent,' '),
                        NVL(im.item_grandparent,' '),
                        im.item_level,
                        im.tran_level,
                        rh.to_dept,
                        rh.to_class,
                        rh.to_subclass,
                        im.dept,
                        im.class,
                        im.subclass
              FROM v_restart_reclass rv,
                        reclass_item ri,
                        reclass_head rh,
                        packitem pi,
                        item_master im
             WHERE rh.reclass_no = ri.reclass_no
                  AND rh.reclass_date <= TO_DATE(:ps_vdate, 'YYYYMMDD')
                  AND im.simple_pack_ind = 'Y'
                  AND (im.item = pi.pack_no OR
                        im.item_parent = pi.pack_no OR
                        im.item_grandparent = pi.pack_no)
                  AND EXISTS (SELECT 'x'
                                        FROM item_master im1
                                       WHERE pi.item = im1.item
                                             AND im1.item_level =
im1.tran_level
                                             AND (ri.item = im1.item
                                                OR ri.item =
im1.item_parent
                                                OR ri.item =
im1.item_grandparent))
                  AND rv.driver_value = rh.reclass_no
                  AND rv.driver_name  = :ps_driver_name
                  AND rv.num_threads  = TO_NUMBER(:ps_num_threads)
                  AND rv.thread_val   = TO_NUMBER(:ps_thread_val)
                  AND (rh.reclass_no > NVL(:ps_restart_reclass_no, -999)
OR
                        (rh.reclass_no = :ps_restart_reclass_no AND
                        ri.item > :ps_restart_item))
```

```
ORDER BY 2, 3;
```

Upload Stock Count Results [stkupld]

Design Overview

The purpose of this batch module is to accept cycle count details from an external system. The cycle count transactions will be compared with Retek system snapshots of stock on hand at the time of the cycle count to determine the stock and/or dollar adjustments to be made. The following common functions will be performed on each stock record read from the input file:

- if record exists on STAKE_SKU_LOC then update it
- if record doesn't exist on STAKE_SKU_LOC
  - validate that item/location exists in system
  - insert a record into STAKE_SKU_LOC
  - insert stock take record into STAKE_SKU_LOC.
- if record is a pack - update/insert information on STAKE_SKU_LOC for all component items

TABLE	SELECT	INSERT	UPDATE	DELETE
stake_qty	No	Yes	No	No
stake_sku_loc	No	Yes	Yes	No
item_loc_soh	Yes	No	No	No
item_loc	Yes	No	No	No
item_master	Yes	No	No	No
wh	Yes	No	No	No
stake_head	Yes	No	No	No
stake_location	Yes	Yes	No	No
stake_prod_loc	Yes	No	No	No
v_packsku_qty	Yes	No	No	No
system_options	Yes	No	No	No

This program reads a user-created interface file of cycle counts. Files will be unique to location and cycle count ID. All records will be validated for layout. Invalid layouts will produce fatal errors. Fields will be validated for content. Invalid contents will produce non-fatal errors. Valid records will update the physical_count_qty field on STAKE_SKU_LOC for a given item/location/ cycle count combination. If the item is a pack, component items will have their component quantity added to the pack_comp_qty field on STAKE_SKU_LOC. If an item does not exist on STAKE_SKU_LOC, the item/location combination will be validated on the item/location tables and a new record will be inserted to STAKE_SKU_LOC.

Fatal errors will terminate file processing. Non-fatal errors will discontinue record processing and will write invalid record to a reject file.

File layout will be verified by interface library routines:

- get_record: validates common fields in file head record and fills structure of remaining fields that are passed from this program
- process_dtl_ftail: called after end-of-file is reached. Will process file trailer record by validating its layout and verifying that the file record counter is set properly.

Re-run:

- If this program terminates normally, restart without recovery.
- If this program terminates abnormally, restart without recovery.

Scheduling Constraints

Processing Cycle:	PHASE 3 (Daily)
Scheduling Diagram:	This program will probably be run at the start of the batch cycle during POS polling, or possibly at the end of the batch run if pending warehouse transactions exist. It can be scheduled to run multiple times throughout the day, as WMS or POS data becomes available.
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

The logical unit of work for the stock take upload module will be a count of discrete inventory transactions. Each record will be uniquely identified by a location and item. The logical unit of work will be defined as a number of these transaction records, determined by the commit_max_ctr field on the restart_control table.

The file records will be grouped in numbers equal to the commit_max_ctr. After all records in a given read are processed (or rejected), the restart commit logic and restart file writing logic will be called, after which the following group of file records will be read and processed. The commit logic will save the current file pointer position in the input file and any application image information (e.g. record and reject counters) and commit all database transactions. The file writing logic will append the temporary holding files to the final output files.

The commit_max_ctr field should be set to prevent excessive rollback space usage and to reduce the overhead of file I/O. The recommended commit counter setting is 10,000 records (subject to change based on experimentation).

Error handling will recognize three levels of record processing: process success, non-fatal errors, and fatal errors. Item level validation will occur on all fields before table processes are initiated. If all field-level validations return successfully, inserts and updates will be allowed. If a non-fatal error is produced, the remaining fields will be validated but the record will be rejected and written to the reject file. If a fatal error is returned, file processing will end immediately. A restart will be initiated from the file pointer position saved in the restart_bookmark string at the time of the last commit point that was reached during file processing.

Program Flow

N/A

Shared Modules

validate_all_numeric: interface library function.

validate_all_numeric_signed: interface library function.

valid_date: interface library function.

Function Level Description

init():

initialize restart recovery

open input file

- file should be specified as input parameter to program

declare final output filename (used in restart_write_file logic)

open reject file (as a temporary file for restart)

- file should be specified as input parameter to program

call restart_file_init logic

assign application image array variables- line counter (g_l_rec_cnt), reject counter (g_l_rej_cnt), cycle_count, stocktake date

if fresh start (l_file_start = 0)

read file header record (get_record)

if (record type <> 'FHEAD') Fatal Error

validate file type = 'STKU'

else fseek to l_file_start location

validate head (validate_head())

validate cycle count id & cycle count date are valid on stake_head

process()

loop - fread rows (equal to commit counter) of input file

if end of file encountered, decrement for loop counter and set end of file flag to true

for loop to process all records read

copy input detail structure elements to stake_sku_loc structure elements

validate elements (validate_detail())

if non-fatal error occurs write detail structure to reject file (write_to_rej_file) and continue at the top of the for-loop

update stake_sku_loc

if record doesn't exist, validate that item/location is valid

if invalid then non-fatal error -write record & continue

insert to stake_sku_loc (if display pack also insert component items)

end loop for loop to process individual records

insert structure of arrays (for valid record counter) into stake_sku_loc

restart file commit - save current input file position, and application image (cnt, cycle count & date)

restart write file function

if end of file reached then break from while loop

end outer loop to read from file

restart commit final

validate_head()

if file type != 'STKU' then fatal file type error

copy stocktake_date, cycle count, location type, location value into variables, nullpad the value and perform necessary validation for each variables.

If the location type is a Store, populate the warehouse variable with –1 and the store value to a variable. If the location type is a Warehouse, populate the store variable with –1 and the warehouse value to a variable. Otherwise, return fatal error.

Open the c_valid_cc cursor to get the stocktake type for each cycle count and the stocktake_date. If there is no data found, return an error message.

validate_detail()

if record type != FDETL then fatal file layout error.

Copy the item type and also the item value into variables. Check the item type.

If the item_type is ITM, then copy the item value into a variable. If the item_type is REF, then get the item_parent, transaction level and item level information from the c_get_item cursor. If the transaction level is equal to the item level, copy the item value into a variable. If the item level is greater than the transaction level, copy the value of the parent item into a variable.

Copy the quantity, location description into variables, nullpad all fields, left shift item and qty, check that store and qty are all numeric, place decimal in qty field.

ON Fatal Error

- Exit Function with -1 return code

ON Non-Fatal Error

- write out rejected record to the reject file using write_to_rej_file functionn, pass pointer to detail record structure, number of bytes in structure, and reject file pointer

I/O Specification

Input File

The input file should be accepted as a runtime parameter at the command line.

Record Name	Field Name	Field Type	Description
File Header	file type record descriptor	Char(5)	hardcode ‘FHEAD’
	file line identifier	Number(10)	Id of current line being processed., hardcode ‘000000001’
	file type	Char(4)	hardcode ‘STKU’
	file create date	Date(14) YYYYMMDDHHMISS	date written by convert program
	stocktake_date	Date(14) YYYYMMDDHHMISS	stake_head.stocktake_date
	cycle count	Number(8)	stake_head.cycle_count
	loc_type	Char(1)	hardcode ‘W’ or ‘S’
	location	Number(10)	stake_location.wh or stake_location.store
Transaction record	file type record descriptor	Char(5)	hardcode ‘FDETL’
	file line identifier	Number(10)	Id of current line being processed, internally incremented
	item type	Char(3)	hardcode ‘ITM’
	item value	Char(25)	item id
	inventory quantity	Number(12,4)	total units or total weight
	location description	Char(30)	Where in the location the item exists. Ex: Back Stockroom or Front Window Display
File trailer	file type record descriptor	Char(5)	hardcode ‘FTAIL’
	file line identifier	Number(10)	Id of current line being processed, internally incremented

Record Name	Field Name	Field Type	Description
	file record count	Number(10)	Number of detail records.

Reject File

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will be created by the interface library routines and the detail records will be created using the write_to_rej_file function. A reject line counter will be kept in the program and is required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and trailer record and the file line count will be equal to 0.

A final reject file name, a temporary reject file name, and a reject file pointer should be declared. The reject file pointer will identify the temporary reject file. This is for the purposes of restart recovery. When a commit event takes place, the restart_write_function should be called (passing the file pointer, the temporary name and the final name). This will append all of the information that has been written to the temp file since the last commit to the final file. Therefore, in the event of a restart, the reject file will be in synch with the input file.

Error File

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

N/A

TSF Subscription API

Functional Area

Transfer Subscription

Design Overview

RMS will expose an API that will allow external systems to create, edit, and delete transfers within RMS. The transaction will be performed immediately upon message receipt so success or failure can be communicated to the calling application.

Transfer messages will be sent across the Retek Integration Bus (RIB). Transfers can be created, modified or deleted at the header or modified or deleted the detail level, each with its own message type.

Consume Module

File name: rmssub_xtsfs/b.pls

```
RMSSUB_XTSF.CONSUME
(O_status_code      IN OUT    VARCHAR2,
O_error_message     IN OUT    RTK_ERRORS.RTK_TEXT%TYPE,
I_message           IN        RIB_OBJECT,
I_message_type      IN        VARCHAR2)
```

This procedure will need to initially ensure that the passed in message type is a valid type for transfer messages. The valid message types for transfer messages are listed in a section below.

If the message type is invalid, a status of "E" should be returned to the external system along with an appropriate error message informing the external system that the status is invalid.

If the message type is valid, the generic RIB_OBJECT needs to be downcast to the actual object using the Oracle's treat function. There will be an object type that corresponds with each message type. If the downcast fails, a status of "E" is returned to the external system along with an appropriate error message informing the external system that the object passed in is invalid.

If the downcast is successful, then consume needs to verify that the message passes all of RMS's business validation. It calls the RMSSUB_XTSF_VALIDATE.CHECK_MESSAGE function to determine whether the message is valid. If the message passed RMS business validation, then the function returns true, otherwise it returns false. If the message fails RMS business validation, a status of "E" is returned to the external system along with the error message returned from the CHECK_MESSAGE function.

Once the message has passed RMS business validation, it is persisted to the RMS database. It calls the RMSSUB_XTSF_SQL.PERSIST() function. If the database persistence fails, the function returns false. A status of "E" is returned to the external system along with the error message returned from the PERSIST() function.

Once the message has been successfully persisted, there is nothing more for the consume procedure to do. A success status, "S", is returned to the external system indicating that the message has been successfully received and persisted to the RMS database.

Business Validation Module

File name: rmssub_xtsfvals/b.pls

```
RMSSUB_XTSF_VALIDATE.CHECK_MESSAGE
(O_error_message      IN OUT       RTK_ERRORS.RTK_TEXT%TYPE,
O_tsf_rec             OUT NOCOPY   RMSSUB_XTSF_SQL.TSF_REC,
I_message             IN           RIB_XTSFDESC_REC,
I_message_type        IN           VARCHAR2)
```

This overloaded function performs all business validation associated with create/modify messages and builds the transfer API record with default values for persistence in the transfer related tables. Any invalid records passed at any time results in message failure.

Like other APIs, the transfer API expects a snapshot of the record on both a header modify and a detail modify message, instead of only the fields that are changed. For a detail modify message, the transfer number, locations and location types will be validated at the header level; all other header fields are ignored. For a header modify message, no details should be included in the message.

All populated fields in the message will be used to create an instance of the RMSSUB_XTSF.TSF_REC object. A number of fields will be defaulted if they are not populated and others will be left as null.

TRANSFER CREATE:

- Check required fields on both header and detail nodes.
- Verify transfer number does NOT already exist.
- Verify attributes in the message header are correct.
- Verify attributes in the message detail are correct.
- Create item/loc relation if not already exist, including creating item_loc_soh, item_supp_country_loc, and price_hist records. If a pack item is involved, these records will be created for all component items.
- Populate record TSF_REC with message data for both header and detail.
- Default the following fields if they are populated in the message:
  - freight_code
  - tsf_type
  - supp_pack_size

TRANSFER MODIFY

- Check required fields on the header node.
- Verify transfer number already exists.
- Verify attributes in the message header are correct.
- Populate record TSF_REC.HEADER with message data.

TRANSFER DETAIL MODIFY

- Check required fields on the detail node.
- Verify transfer/items/inventory statuses already exist.
- Check that the new transfer quantity is not less than the current distribution, selected or shipped quantity, the quantity that has currently been processed, for that item on the transfer.
- Populate record ORDER_REC.TSF_DETAIL with message data.

RMSSUB_XORDER_VALIDATE.CHECK_MESSAGE

(O_error_message	IN OUT	RTK_ERRORS.RTK_TEXT%TYPE,
O_tsf_rec	OUT NOCOPY	RMSSUB_XTSF_SQL.TSF_REC,
I_message	IN	RIB_XTSFREF_REC,
I_message_type	IN	VARCHAR2)

This overloaded function performs all business validation associated with delete messages and builds the transfer API record with default values for persistence in the transfer related tables. Any invalid records passed at any time results in message failure.

TRANSFER DELETE

- Check required fields.
- Verify transfer number already exists.
- Verify that transfer is not already in progress.
- Populate record TSF_REC.HEADER with the transfer number and locations for delete.

TRANSFER DETAIL DELETE

- Check required fields.
- Verify transfer/item/inventory status already exists.
- Verify that transfer detail is not already in progress
- If any of the following have values, the detail is in progress: distribution qty, selected qty, received qty, shipped qty, or cancelled qty.
- Populate record TSF_REC.TSF_DETAIL with the transfer/item/quantities for delete.
- If this is the last detail for this transfer, delete the header record as well.

Bulk or single DML module

Filename: rmssub_xtsfsqls/b.pls

```
RMSSUB_XTSF_SQL.PERSIST
(O_error_message      IN OUT   RTK_ERRORS.RTK_TEXT%TYPE,
I_tsf_rec             IN       RTMSSUB_XTSF_SQL.TSF_REC,
I_message_type        IN       VARCHAR2)
```

This function checks the message type to route the object to the appropriate internal functions that perform DML insert, update and delete processes.

TRANSFER CREATE

- Inserts records in the TSFHEAD, TSFDETAIL, TSFDETAIL_CHRG tables.
- Update ITEM_LOC_SOH records for all item/location combinations and all pack comp/location combinations.

TRANSFER MODIFY

- Updates a record in the TSFHEAD table.

TRANSFER DELETE

- Delete an order from TSFHEAD, TSFDETAIL, TSFDETAIL_CHRG tables.
- Update ITEM_LOC_SOH records for all item/location combinations and all pack comp/location combinations.

TRANSFER DETAIL MODIFY

- Updates records in the TSFDETAIL table.
- Update ITEM_LOC_SOH records for all item/location combinations and all pack comp/location combinations.

TRANSFER DETAIL DELETE

- Delete records from TSFDETAIL table.
- Update ITEM_LOC_SOH records for all item/location combinations and all pack comp/location combinations.

Message DTD

Here are the filenames that correspond with each message type. Please consult the mapping documents for each message type in order to get a detailed picture of the composition of each message.

Message Types	Message Type Description	Document Type Definition (DTD)
XTsfCre	Transfer Create Message	XTsfDesc.dtd
XTsfMod	Transfer Modify Message	XTsfDesc.dtd
XTsfDel	Transfer Delete Message	XTsfRef.dtd
XTsfDtlMod	Transfer Detail Modify Message	XTsfDesc.dtd

Message Types	Message Type Description	Document Type Definition (DTD)
XTsfDtlDel	Transfer Detail Delete Message	XTsfRef.dtd

Design Assumptions

Required fields are shown in mapping documents.

Tables

TABLE	SELECT	INSERT	UPDATE	DELETE
TSFHEAD	YES	YES	YES	YES
TSFDETAIL	YES	YES	YES	YES
TSFDETAIL_CHRG	NO	YES	YES	YES
ITEM_LOC_SOH	YES	YES	YES	NO
ITEM_LOC	YES	YES	NO	NO
WH	YES	NO	NO	NO
STORE	YES	NO	NO	NO
ITEM_MASTER	YES	NO	NO	NO
DEPS	YES	NO	NO	NO