Retek® Merchandising System™ 11.0.5

Operations Guide Addendum

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

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

Chapter 1 – Introduction

This addendum to the RMS 11 Operations Guide presents changes that have resulted from work completed during RMS 11.0.5 development. The RMS 11 Operations Guide volumes impacted include:

- Volume 1, Functional Overviews
- Volume 2, Message Publication and Subscription Designs
- Volume 4, Batch Designs

Chapter 2 – Vendor subscription functional overview and design

Functional overview

RMS subscribes to vendor information that is published from an external financial application. ‘Vendor’ refers to either a partner or a supplier. Vendor information includes partner, supplier, and supplier address data. The financial application (such as Oracle Financials) sends the information to RMS via the RIB.

Processing includes a check for the appropriate financial application in RMS on the SYSTEM_OPTIONS table’s financial_ap column.

Processing includes a check for oracle_financials_vers on the SYSTEM_OPTIONS table. If RMS determines that the version of Oracle Financials is 11.5.10, RMS processes the information differently due to the requirements of Oracle Financials 11.5.10.

Both partners and suppliers bill retailers for their work. Partners provide retailers with services, such as transportation of goods, escheatment, providing credit, and so on. Suppliers provide retailers with merchandise items or other goods.

Vendor subscription API

Functional Area

Supplier

Business Overview

Data Flow:

An external system will publish a supplier, thereby placing the supplier information onto the RIB (Retek Information Bus). RMS will subscribe to the supplier information as published from the RIB and place the information onto RMS tables depending upon the validity of the records enclosed within the message.

Message Structure:

The Supplier message is a hierarchical message that will consist of a supplier header record, a series of address records under the header record.

The header record will contain information about the supplier as a whole. The address records will identify the addresses associated with the supplier.

Package Impact

Subscribing to a supplier message entails the use of one public consume procedure. This procedure corresponds to the type of activity that can be done to a supplier record (in this case create/update).

Filename: rmssub_vendorcres/b.pls

Public API Procedures

```
RMSSUB_VENDORCRE.CONSUME
(O_status_code          IN      OUT  VARCHAR2,
O_error_message         IN      OUT  VARCHAR2,
I_message               IN      CLOB);
```

This procedure accepts a XML file in the form of an Oracle CLOB data type from the RIB (I_message). This message will contain a supplier message consisting of the aforementioned header and detail records. The procedure will then place a call to the main RMSSUB_SUPPLIER.CONSUME function in order to validate the XML file format and, if successful, parse the values within the file through a series of calls to RIB_XML. The values extracted from the file will then be passed on to private internal functions, which will validate the values and place them on the supplier and address tables depending upon the success of the validation.

Private Internal Functions and Procedures (rmssub_vendorcre.pls):

Error Handling:

If an error occurs in this procedure, a call will be placed to HANDLE_ERRORS in order to parse a complete error message and pass back a status to the RIB.

```
HANDLE_ERRORS
(O_status           IN OUT      VARCHAR2,
IO_error_message    IN OUT      VARCHAR2,
I_cause             IN          VARCHAR2,
I_program           IN          VARCHAR2);
```

This function is used to put error handling in one place in order to make future error handling enhancements easier to implement. All error handling in the internal RMSSUB_SUPPLIER package and all errors that occur during subscription in the RMSSUB_VENDORCRE package (and whatever packages it calls) will flow through this function.

The function consists of a call to API_LIBRARY.HANDLE_ERRORS. API_LIBRARY.HANDLE_ERRORS accepts a program name, the cause of the error and potentially an unparsed error message if one has been created through a call to SQL_LIB.CREATE_MESSAGE. The function uses these input variables to parse a complete error message and pass back a status, depending upon the message and error type, back up through the consume function and up to the RIB.

Private Internal Functions and Procedures (other):

All of the following functions exist within RMSSUB_SUPPLIER.

Main Consume Function:

```
RMSSUB_SUPPLIER.CONSUME
(O_status           OUT     VARCHAR2,
O_error_message     OUT     VARCHAR2,
I_document          IN      CLOB);
```

This procedure accepts a XML file in the form of an Oracle CLOB data type from the RIB (I_message) from the aforementioned public vendor procedure whenever a message is made available by the RIB. This message will consist of the aforementioned header and detail records.

The procedure will first get the financial_ap value from the system options table. It will then validate the XML file format and, if successful, call internal functions to parse the values within the file through a series of calls to RIB_XML. The values extracted from the file will then be passed on to private internal functions, which will validate the values and place them on the appropriate supplier tables depending upon the success of the validation. Supplier Address records will contain at least one detail record consisting of an Oracle Vendor Site Id, an Organizational Unit Id, and an Address Key. The procedure will loop through the available detail records calling PARSE_ADDRESS and then PROCESS_ADDRESS to validate the given information and place it on the appropriate tables. The procedure will then call the CHECK_ADDR function to check that the proper addresses have been associated with the supplier.

XML Parsing

```
PARSE_SUPPLIER
(O_error_message      OUT        VARCHAR2,
O_table_locked        IN OUT     BOOLEAN,
O_supplier_record     OUT        sups%ROWTYPE,
I_supplier_root       IN OUT     xmldom.DOMElement);
```

This function will used to extract the header level information from the Supplier XML file and place that information onto an internal Supplier header record.

Record is based upon the supplier table:

SUPS%ROWTYPE;

```
PARSE_ADDRESS
(O_error_message      OUT      VARCHAR2,
O_table_locked        OUT      BOOLEAN,
O_address_record      OUT      addr%ROWTYPE,
I_addr_node           IN OUT   xmldom.DOMElement);
```

This function will used to extract the address level information from the Supplier XML file and place that information onto an internal address record.

Record is based upon the address table:

ADDR%ROWTYPE;

Validation:

```
PROCESS_SUPPLIER
(O_error_message    IN OUT   VARCHAR2,
O_table_locked      IN OUT   BOOLEAN,
IO_supplier_record  IN OUT   sups%ROWTYPE)
```

After the values are parsed for a particular supplier record, RMSSUB_SUPPLIER.CONSUME will call this function, which will in turn call various functions inside RMSSUB_SUPPLIER in order to validate the values and place them on the appropriate supplier table depending upon the success of the validation. Either INSERT_SUPPLIER or UPDATE_SUPPLIER is called to actually insert or update the supplier table.

```
PROCESS_ADDRESS
(O_error_message    OUT      VARCHAR2,
O_table_locked      OUT      BOOLEAN,
I_supplier_no       IN       sups.supplier%TYPE,
I_address_record    IN       addr%ROWTYPE)
```

After the values are parsed for a particular address record, RMSSUB_SUPPLIER.CONSUME will call this function. If the financial_ap system option is set to 'O', this function will call various functions inside RMSSUB_SUPPLIER in order to validate the values and place them on the appropriate address table depending upon the success of the validation. Either INSERT_ADDRESS or UPDATE_ADDRESS is called to actually insert or update the address table.

```
INSERT_SUPPLIER
(O_error_message    IN     OUT VARCHAR2,
I_supplier_record   IN     sups%ROWTYPE);
```

This function will first check the unit_options table to see what the value of dept_level_orders is. If the dept_level_orders value is ‘Y’, then the inv_mgmt_lvl will be defaulted to ‘D’. If the dept_level_orders value is anything other than ‘Y’, the inv_mgmt_lvl will be set to ‘S.’

The function then takes the information from the passed-in supplier record and inserts it into the sups table.

```
FUNCTION UPDATE_SUPPLIER
(O_error_message    IN OUT    VARCHAR2,
O_table_locked      IN OUT    BOOLEAN,
I_supplier_record   IN        sups%ROWTYPE)
```

This function will update the sups table using the values contained in the I_supplier_record.

```
FUNCTION UPDATE_ADDRESS
(O_error_message    IN OUT    VARCHAR2,
O_table_locked      IN OUT    BOOLEAN,
I_supplier_no       IN        sups.supplier%TYPE,
I_address_record    IN        addr%ROWTYPE);
```

Updates the supplier information to the address table.

```
CHECK_CODES
(O_error_message    IN OUT        VARCHAR2,
I_record_variable   IN            VARCHAR2,
I_code_value        IN            VARCHAR2);
```

The RMSSUB_SUPPLIER package, specifically the functions check_codes() and check_fkeys(), will send back descriptive error messages when codes are not valid or if a foreign key constraint is violated.

```
INSERT_ADDRESS
(O_error_message    IN OUT        VARCHAR2,
O_table_locked      OUT           BOOLEAN,
I_supplier_no       IN            sups.supplier%TYPE,
I_address_record    IN            addr%ROWTYPE);
```

Insert supplier information to address table. If the address in the passed-in address record is the primary address for a particular supplier/address type, this function will update the current primary address so that it is no longer the primary.

```
VALIDATE_SUPPLIER_RECORD
(O_error_message    IN OUT    VARCHAR2,
I_supplier_record   IN        sups%ROWTYPE);
```

Validate that all the necessary records are populated.

```
VALIDATE_ADDRESS_RECORD
(O_error_message    IN OUT    VARCHAR2,
I_address_record    IN        addr%ROWTYPE,
I_supplier_no       IN        sups.supplier%TYPE);
```

Validate that all the necessary records are populated.

```
CHECK_NULLS
(O_error_message  IN OUT    VARCHAR2,
I_record_variable IN        VARCHAR2,
I_record_name     IN        VARCHAR2);
```

This validation is used for both SUPPLIERS and PARTNERS. This function will check that the passed-in record variable is not null. If it is, it will return an error message.

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)
VendorCre	Vendor Create Message	VendorDesc.dtd

Design Assumptions

- One of the primary assumptions in the current API approach is that ease of code will outweigh performance considerations. It is hoped that the 'trickle' nature of the flow of data will decrease the need to dwell on performance issues and instead allow developers to code in the easiest and most straight forward manner.
- The adaptor is only setup to call stored procedures, not stored functions. Any public program then needs to be a procedure.

Tables

TABLE	SELECT	INSERT	UPDATE	DELETE
SUPS	Yes	Yes	Yes	No
ADDR	Yes	Yes	Yes	No
SYSTEM_OPTIONS	Yes	No	No	No
UNIT_OPTIONS	Yes	No	No	No
CODE_DETAIL	Yes	No	No	No

Chapter 3 – Batch designs

New and changed upload from supplier [ediupcat] batch design

Design Overview

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

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

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

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

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

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

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

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

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

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

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

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DIFF_GROUP_DETAIL	No	Yes	No	No	No
DIFF_IDS	No	Yes	No	No	No
EDI_NEW_ITEM	No	Yes	Yes	Yes	No
EDI_COST_CHG	No	No	Yes	Yes	No
EDI_COST_LOC	No	No	Yes	Yes	No
ITEM_LOC	No	Yes	No	No	No
ITEM_MASTER	No	Yes	No	No	No
ITEM_SUPP_COUNTRY	No	Yes	No	No	No
ITEM_SUPP_COUNTRY_BRACKET_COST	No	Yes	No	No	No
ITEM_SUPP_COUNTRY_LOC	No	Yes	No	No	No
ITEM_SUPPLIER	No	Yes	No	No	No
ITEM_ZONE_PRICE	No	Yes	No	No	No
PACKITEM	No	Yes	No	No	No
PERIOD	No	Yes	No	No	No
STORE	No	Yes	No	No	No
SUP_BRACKET_COST	No	Yes	No	No	No
SUPS	No	Yes	No	No	No
WH	No	Yes	No	No	No

Scheduling Constraints

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

Restart Recovery

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

Program Flow

N/A

Shared Modules

SQL_LIB.BATCH_MSG—to write error messages

CURRENCY_SQL.CONVERT_BY_LOCATION—convert unit cost and unit retail

COUNTRY_VALIDATE_SQL.EXISTS_ON_TABLE—validates origin_country_id

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

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

Function Level Description

init()

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

process()

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

format_FDETL():

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

validate_FDETL()

- Validate that the input file has at least one of the item, VPN and ref_item fields populated. If none of the above fields exists, issue an error message and return NON_FATAL.
- Validate break to sell indicator. Values should only be in 'Y','N' and null. . If none of the stated values exists, issue an error message, reject the record and return NON_FATAL.
- Validate supplier by calling validate_supplier.
- When item parent passed in from the input file is not null, call validate_item_parent.
- Call validate_parent_VPN to validate that the parent VPN exists in the system and find the parent_item according to the parent_vpn.
- If both item parent and parent VPN are not null, compare the input item parent with the item parent retrieved from function validate_parent_VPN, if they are differnt, log an error and return NON_FATAL. Otherwise, if the input item parent is null, the item parent retrieved from function validate_parent_VPN should be used.
- Call functions validate_origin_country_id and validate_uom.
- When both ref_item_type and ref_item are presented, call function check_ref_item and passing the ref_item and ref_item_type to the function.
- If the item field has value,
  - call function validate_item;
  - if the item does not equal ref_item, call function validate_ref_item;
  - if the item VPN is not null, call validate_vpn.
- If the record's item field does not have a value, process as follows:
  - Call get_item
- If item parent is not null, item parent description or item parent number type is null, call function get_item_info(). Pass in item parent number and variables to hold the item_parent_desc and item_parent_number_type. Note dummy variables are needed to hold other parameters.
- If both VPN and ref_item are not null and their corresponding item exists in RMS, call function validate_VPN_vs_ref_item to make sure that the item is not above the transaction level. Since an item that is above the transaction level could not have a ref_item. If the function call doesn't return true, return whatever the function returns.
- If the case_ref_item field is not null, call function process_case.
- Validate that the value for break_to_sell_ind. If the value for the column is other than 'Y','N' or null, return NON-FATAL to reject the record. An error message should be written to the error file to state the reject reason.
- If the item_diff field is not null, call function validate_diffs.
- Validate item_level. Value for the field must not be null and should be greater than zero. Otherwise, return NON-FATAL to reject the record. An error message should be written to the error file to state the reject reason.

- Validate tran_level. Value for the field must not be null and should be greater than zero. Otherwise, return NON-FATAL to reject the record. An error message should be written to the error file to state the reject reason.

validate_supplier():

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

validate_item()

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

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

validate_ref_item()

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

check_ref_item()

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

validate_parent_VPN()

Validate the parent_VPN against the item_supplier and edi_new_item tables.

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

validate_origin_country_id()

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

validate_uom()

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

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

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

validate_each_uom()

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

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

validate_vpn()

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

Validate_item_parent()

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

Find_ item_by_ref_item()

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

get_item()

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

Get_item_info()

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

convert_currency()

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

validate_diffs():

- Check if diffs are consecutive. Otherwise, return a NON-FATAL error.
- Check if there are duplicate diffs. Raise a NON-FATAL error if two diffs are the same.
- If item_diff_1 is not null
    - Open c_diff_1 cursor to validate if the item diff is existing in the table. If item_diff is found successfully, return true. If it doesn't have value, log an error message to state so and return NON-FATAL. This record will then be rejected.
    - If item_level passed is greater than 1, open c_parent_diff_1. Validate if the item diff is existing in the table. If item_diff is found successfully, return true. If it doesn't have value, log an error message to state so and return NON-FATAL. This record will then be rejected.
- If item_diff_2is not null
    - Open c_diff_2 cursor to validate if the item diff is existing in the table. If item_diff is found successfully, return true. If it doesn't have value, log an error message to state so and return NON-FATAL. This record will then be rejected.
    - If item_level passed is greater than 1, open c_parent_diff_2. Validate if the item diff is existing in the table. If item_diff is found successfully, return true. If it doesn't have value, log an error message to state so and return NON-FATAL. This record will then be rejected.
- If item_diff_3 is not null
    - Open c_diff_3 cursor to validate if the item diff is existing in the table. If item_diff is found successfully, return true. If it doesn't have value, log an error message to state so and return NON-FATAL. This record will then be rejected.
    - If item_level passed is greater than 1, open c_parent_diff_3. Validate if the item diff is existing in the table. If item_diff is found successfully, return true. If it doesn't have value, log an error message to state so and return NON-FATAL. This record will then be rejected.
- If item_diff_4 is not null
    - Open c_diff_4 cursor to validate if the item diff is existing in the table. If item_diff is found successfully, return true. If it doesn't have value, log an error message to state so and return NON-FATAL. This record will then be rejected.
    - If item_level passed is greater than 4, open c_parent_diff_1. Validate if the item diff is existing in the table. If item_diff is found successfully, return true. If it doesn't have value, log an error message to state so and return NON-FATAL. This record will then be rejected.

process_item()

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

insert_new_item()

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

update_item_info()

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

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

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

Validate_VPN_vs_ref_item():

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

process_case()

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

final()

- restart/recovery close, close files

I/O Specification

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

FHEAD file header

FDETL item info

FTAIL file trailer

Input Files

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

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

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

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

Record Name	Field Name	Field Type	Default Value	Description
	Case lwh UOM	Char(4)		Case dimension unit of measure.
	Case liquid volume	Number(12)		Case liquid volume or capacity (4 implied decimal places)
	Case liquid volume UOM	Char(4)		Unit of measure of the case liquid volume/capacity
	Gross pallet weight	Number(12)		Gross pallet weight (4 implied decimal places)
	Net pallet weight	Number(12)		Net pallet weight (4 implied decimal places)
	Pallet weight UOM	Char(4)		Unit of measure of the pallet weight
	Pallet length	Number(12)		Pallet length (4 implied decimal places)
	Pallet width	Number(12)		Pallet width (4 implied decimal places)
	Pallet height	Number(12)		Pallet height (4 implied decimal places)
	Pallet lwh UOM	Char(4)		Pallet dimension unit of measure.
	Ti	Number(12)		Shipping units (cases) in one tier of a pallet (4 implied decimal places)
	Hi	Number(12)		Number of tiers in a pallet (height). (4 implied decimal places)
	Pack Size	Number(12)		Supplied pack size. I.e., Number of eaches per case pack. This is the quantity that orders must be placed in multiples of for the supplier for the item.
	Inner pack size	Number(12)		Supplied inner pack size. I.e., Number of eaches per inner container.
	Origin Country ID	Char(3)		Supplied origin country ID.
	Standard UOM	Char(4)		Unit of measure in which stock of the item is tracked at a corporate level.

Record Name	Field Name	Field Type	Default Value	Description
	UOM Conversion Factor	Number(20)		Conversion Factor, 10 implied decimal places. Conversion factor between an "Each" and the standard_uom when the standard_uom is not in the quantity class (e.g. if standard_uom = lb and 1 lb = 10 eaches, this factor will be 10). This factor will be used to convert sales and stock data when an item is retailed in eaches but does not have eaches as its standard unit of measure.
	Packing Method	Char(6)		Packing Method code (HANG,FLAT)
	Location	Number(10)		RETEK location that the supplier distributes to or this may be a number used by the supplier to identify a non-RETEK location.
	Location Type	Char(1)		This field will contain the type of location (‘S’ for store and ‘W’ for warehouse).
	Bracket Value 1	Number (12,4)		This will contain the primary bracket value of the supplier.
	Bracket UOM 1	Char(4)		This field will contain the unit of measure of the primary bracket.
	Bracket Type 1	Char (6)		This field will contain the UOM class.
	Bracket Value 2	Number (12,4)		This will contain the secondary bracket value for the supplier.
	Unit cost new	Number (20,4)		This field will contain the new unit cost of the bracket.
	Case Bracket Value 1	Number (12,4)		This will contain the primary bracket value of the supplier for a case UPC.
	Case Bracket UOM 1	Char(4)		This field will contain the unit of measure of the primary bracket for a case UPC.
	Case Bracket Type 1	Char (6)		This field will contain the UOM class for a case UPC.
	Case Bracket Value 2	Number (12,4)		This will contain the secondary bracket value for the supplier for a case UPC.
	Case Unit cost new	Number (20,4)		This field will contain the new unit cost of the bracket for a case UPC.

Record Name	Field Name	Field Type	Default Value	Description
	Item_diff_1	Char(10)		This field will hold a unique number (identifier) of the differential types. (for example, diff_type = 'S' might have these ids: 1, 50, 1000; then diff_type= 'C' cannot use the same numbers; the ids will have to be different: 2, 20,51, 1001)
	Item_diff_2	Char(10)		As above
	Item_diff_3	Char(10)		As above
	Item_diff_4	Char(10)		As above
	unit_retail	Number(20,4)		This field contains the suppliers recommended retail value for the item.
	retail_zone_group	Number(4)		This field contains the retail zone group number of the given item.
	consignment_rate	Number(12,4)		This field contains the consignment rate for this item for the supplier.
	break_to_sell_ind	Char(1)		Indicates whether item is a break to sell item.
	Item_level	Char(1)		Indicates the items' item_level
	Tran_level	Char(1)		Indicates the items' tran_level
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being created for output file.
	File Record Counter	Numeric ID(10)		Number of records/transactions processed in current file (only records between head & tail)

Test Conditions

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

Technical Issues

1. Unit retail and cost will be inserted into edi_new_item table for new items only.
2. We are not using permanent substitutions.
3. It is assumed that currency will be in the supplier's currency. This currency must be converted to primary currency for the edi_new_item table. No translation is necessary for the edi_cost_change table since that stores the supplier's currency.

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

POS upload [posupld] batch design

Design Overview

The purpose of this batch module is to process sales and return details from an external point of sale system. The sales/return transactions will be validated against Retek item/store relations to ensure the sale is valid, but this validation process can be eliminated if the sales being passed in have already been screened by sales auditing. The following common functions will be performed on each sales/return record read from the input file:

- read sales/return transaction record
- lock associated record in RMS
- validate item sale
- check if VAT maintenance is required, if so determine the VAT amount for the sale
- write all financial transactions for the sale and any relevant markdowns to the stock ledger.
- post item/location/week sales to the relevant sales history tables
- if a late posting occurs in a previous week (i.e. not in the current week), if the item for which the late posting occurred is forecastable, the last_hist_export_date on the item_loc_soh table has to be updated to the end of week date previous to the week of the late posting. This will result in the sales download interface programs extracting the week(s) for which the late transactions were posted to maintain accurate sales information in the external forecasting system.

Stored Procedures / Shared Modules (Maintainability)

validate_all_numeric: intrface library function.

validate_all_numeric_signed: intrface library function.

valid_date: intrface library function.

PM_API_SQL. GET_RPM_SYSTEM_OPTIONS: called from init(), returns complex_promo_allowed_ind to set pi_multi_prom_ind

CAL_TO_CAL_LDOM database procedure called from get_eow_eom_date() function

CAL_TO_454_LDOM database procedure called from get_eow_eom_date() function

VAT_SQL.GET_VAT_RATE: called from pack_check(), fill_packitem_array() returns the composite vat rate for a packitem.

CURRENCY_SQL.CONVERT: returns the converted monetary amount from

Currency to currency.

NEW_ITEM_LOC: called from item_check(), item_check_orderable(), pack_check_orderable() and pack_check(), creates a new item if one doesn’t already exist for the item/location passed in.

UPDATE_SNAPSHOT_SQL.EXECUTE: called from update_snapshot(), updates the stake_sku_loc and edi_daily_sales tables for late transactions. If the item is a return, edi_daily_sales will not be updated.

NEXT_ORDER_NO: called from consignment_data(), returns the next available generated order number.

STKLDGR_SQL.TRAN_DATA_INSERT: called from consignment_data(), performs tran_data inserts (tran_type 20) for a consignment transaction.

DATES_SQL.GET_EOW_DATE: called from get_eow_eom_date(), returns eow and eom dates.

UOM_SQL.CONVERT: called from validate_THEAD(), converts selling uom to standard uom.

SUPP_ATTRIB_SQL.GET_SUP_PRIMARY_ADDR: called from invc_data(), returns primary supplier address.

INVC_SQL.NEXT_INVC_ID: called from invc_data(), returns invoice_id

PRICING_ATTRIB_SQL.GET_BASE_ZONE_RETAIL(), called from get_loc_item_retail(), returns base zone retail from RPM.

Posupld and VAT:

There are three different data sources in POSUPLD.

- the input file
- RMS stock ledger tables (tran_data in this context)
- RMS base tables (other that stock ledger)

Each of these data sources can be vat inclusive or vat exclusive.

There are five different system variables that are used to determine whether of not the different inputs are vat inclusive or vat exclusive.

- system_options.vat_ind (assume Y for this document)
- system_options.class_level_vat_ind
- system_options.stkldgr_vat_incl_retl_ind
- class.class_vat_ind
- store.vat_include_ind (this is retrieved from the table when RESA is on and read from the input file when RESA is off)

Given the three different data source and all combinations of vat inclusive or vat exclusive, we are left with the 8 potential combinations of inputs to POSUPLD.

Possible POSUPLD inputs			
Scenario	**File**	**RMS**	**Stock Ledger**
1	Y	Y	Y
2	Y	Y	N
*3**	*Y*	*N*	*Y*
*4**	*Y*	*N*	*N*
5	N	Y	Y
6	N	Y	N
7	N	N	Y
8	N	N	N

* Scenarios 3 and 4 are not possible – the file will never have vat when RMS does not.

The combinations of system variables and the resulting scenarios				
System_options Class_level_vat_ind	**System_options Stkldgr vat ind**	**Class Class_vat_ind**	**Store Vat_include_ind**	**Resulting Scenario**
Y	Y	Y	Y - Ignored	1
Y	Y	Y	N - Ignored	1
Y	Y	N	Y - Ignored	7
Y	Y	N	N - Ignored	7
Y	N	Y	Y - Ignored	2
Y	N	Y	N - Ignored	2
Y	N	N	Y - Ignored	8
Y	N	N	N - Ignored	8
N	Y	Y – Ignored	Y	1
N	Y	Y – Ignored	N	5
N	Y	N – Ignored	Y	1
N	Y	N – Ignored	N	5
N	N	Y – Ignored	Y	2
N	N	Y – Ignored	N	6

The combinations of system variables and the resulting scenarios				
N	N	N – Ignored	Y	2
N	N	N – Ignored	N	6

POSUPLD table writes

Scenario 1:

- tran code 1 from file retail.
- tran code 2 from file retail with vat removed.
- retail from file is compared directly with price_hist for off retail check.

Scenario 2:

- tran code 1 from file retail with vat removed.
- tran code 2 not written.
- retail from file is compared directly with price_hist for off retail check.

Scenario 5:

- tran code 1 from file retail with vat added.
- tran code 2 from file retail.
- retail from file has vat added for compare with price_hist for off retail check.

Scenario 6:

- tran code 1 from file retail.
- tran code 2 not written.
- retail from file has vat added for compare with price_hist for off retail check.

Scenario 7:

- tran code 1 from file retail with vat added.
- tran code 2 from file retail.
- retail from file is compared directly with price_hist for off retail check.

Scenario 8:

- tran code 1 from file retail.
- tran code 2 not written.
- retail from file is compared directly with price_hist for off retail check.

Function Level Description

main()

standard Retek main function that calls init(), process(), and final()

init()

initialize restart recovery

open input file (posupld)

- file should be specified as input parameter to program

fetch system variables, including the SYSTEM_OPTIONS.CLASS_LEVEL_VAT_IND.

fetch pi_multi_prom_ind from RPM interface

retrieve all valid promotion types and uom class types

fetch uom class types for look up during THEAD processing

declare memory required for all arrays setup for array processing

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

open lock 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), lock reject file counters (pl_lock_cnt, pl_lock_dtl_cnt), store, transaction_date

if fresh start (l_file_start = 0)

read file header record (get_record)

write FHEAD to lock reject file

if (record type <> ‘FHEAD’) Fatal Error

validate file type = ‘POSU’

else fseek to l_file_start location

validate location and date are valid

set restart variables to ones from restart image

file_process()

This function will perform the primary processing for transaction records retrieved from the input file. It will first perform validation on the THEAD record that was fetched. If the transaction was found to be invalid, a record will be written to the reject file, a non-fatal error will be returned, and the next transaction will be fetched.

Next, the unit retail from price_hist will be fetched by calling the get_unit_retail() function. The retail retrieved from this function will be compared with the actual retail sent in from the input file to determine any discrepancies in sale amounts.

Fetch all of the TDETL records that exist for the transaction currently being processed until a TTAIL record is encountered. Perform validation on the transaction detail records. If a detail record is found to be invalid, the entire transaction will be written to the reject file, a non-fatal error will be returned, and the next record will be fetched. If a valid promotion type (code for mix & match, threshold promotions, etc.) was included in the detail record and it is not an employee disc record, write a record to the daily_sales_discount table. If it is an employee discount record write an employee discount record to tran_data. Finally, accumulate the discount amounts for all transaction detail records for the current transaction, unless the record was an employee discount. Next, establish any vendor funding of promotions. This information is expressed as a percentage of the allowed discount and is retrieved by querying the rpm_promo_xxx tables for the promotion_id and component_id. If the promotion type is 9999 (i.e., all promotion types), call get_deal_contribs to append to pr_deals_contribs arrays zero or more lines of deal and vendor contribution information for the current item

Call the item_process() function to perform item specific processing. Once all records have been processed, write FTAIL record to lock reject file and call posting_and_restart to commit the final records processed since the last commit and exit the function.

item_process()

Check to see if any validation failed for the item before this function was called. If a lock error was found, call write_lock_rej() then return. If an other error was found, call write_rej() and process_detail_error() then return.

Set the item sales type for the current transaction. Valid sales types are ‘R’egular sales, ‘C’learance sales, and ‘P’romotional sales. These will be used when populating the sales types for the item-location history tables. If an item is both on promotion and clearance, and the promotion price is less than the clearance price, than the transaction will be written as a promotion transaction, otherwise as a clearance transaction.

If the system’s VAT indicator is turned to on, VAT processing will be performed. The function vat_calc() will retrieve the vat rate and vat code for the current item-location. The total sales including and excluding VAT will be calculated for use in writing transaction data records. If any VAT errors occur, the entire transaction will be written to the reject file, a non-fatal error will be returned, and the next record will be fetched. A record will be written to vat_history for the item, location, transaction date.

Calculate the item sales totals (i.e. total retail sold, total quantity sold, total cost sold, etc.). If VAT is turned on in the system, calculate exclusive and inclusive VAT sales totals.

Calculate any promotional markdowns that may exist by calling the calc_prom_totals() function. The markdown information calculated here will be used when writing tran_data (tran_type 15) records for promotional markdowns.

Calculate the over/under amount the item was sold at compared to its price_hist record. (The complex_promo_allowed_ind indicator is retrieved from RPM by calling PM_API_SQL.GET_SYSTEM_OPTIONS.) Since we do not create price_hist records of type 9 (promotional retail change) when the complex_promo_allowed_ind = 'Y', we do not know what the promotional retail for this item is. Therefore, we will take the total sales reported from the header record plus the total of sales discounts reported in the TDETL records, divided by the total sales quantity for the item to calculate its unit retail. If the complex_promo_allowed_ind = 'N', we can do a comparison of the price_hist record and the unit retail (total retail / total sales) inputted from the POS file. Any difference using either method will write to the daily_sales_discount table with a promotion type of 'in store' and tran_data (tran_type 15) If the transaction is a return, no daily_sales_discount record will be written, and tran_data records will be written as opposite of what they were sold as (i.e. if the sale was written as a markup, which would be written as a negative retail with a tran_data 15, the return would be written as a 15 with a positive retail).

If the item is a packitem and the transaction is a Sale, the process_pack() function will update the last_hist_export_date field on the item_loc_soh table to the transaction date and the item_loc_hist table will be updated with the transaction information.

If the item currently being processed is a packitem, calculate the retail markdown the item takes for being included in the pack and write a transaction data record as a promotional markdown. This markdown is calculated by comparing the retail contribution of the packitem's component item to the packitem to the component item's regular retail found on the price_hist table. The retail contribution for a component item is calculated by taking the component item's unit retail from price_hist, divided by the total retail of all component items in the packitem, and multiplying the packitem's unit retail. So if the retail contribution of a component item within packitem A is $10, and the same component item's price_hist record has a retail of $14, and there is only one packitem sold, and this component item has a quantity of one, a tran_data

Record (tran_type 15) will be written for $4 (assume no vat is used).

Write transaction data records for sales and returns. If the transaction is a sale, write a tran_data record with a transaction code of 1 with the total sales. If the system VAT indicator is on and the system_options.stkldgr_vat_incl_retl_ind is on, write a tran_data record with a transaction code of 2 for VAT exclusive sales. If the transaction is a return, write a tran_data record (tran_type 1) with negative quantities and retails for the amount of the return. If the system VAT indicator is on and the system_options.stkldgr_vat_incl_retl_ind is on, write a tran_data record (tran_type 2) and negative quantities and retails for the VAT exclusive return. Also, write a tran_data record with a transaction code of 4 for the total return. Any tran_data record that is written should be either VAT exclusive or VAT inclusive, depending on the system_options.stkldgr_vat_incl_retl_ind. If it is set to 'Y', all tran_data retails should be VAT inclusive. If it is set to 'N', all tran_data retails should be VAT exclusive. When writing tran_data records for packitems, always break them down to the packitem level, writing the retail as the packitem multiplied by the component item's price ratio. The packitem itself should never be inserted into the tran_data table.

If the transaction is late (transaction date is before the current date) and it is not a drop shipment, call update_snapshot() to update the stake_sku_loc and edi_daily_sales tables. If the transaction is current, update the edi_daily_sales table only (stake_sku_loc will be updated in a batch program later down the stream). The edi_daily_sales table should only be updated if the items supplier edi sales report frequency = 'D'.

If VAT is turned on in the system, write a record to the vat_history table to record the vat amount applied to the transaction. The VAT amount is calculated by taking the sales including VAT minus the sales excluding VAT.

Update the sales history tables for non-consignment items that are Sale transactions. Do not update for returns. Also, update stock count on the item-location table for Sales and Returns unless the item is on consignment or is drop shipped.

If the dropship indicator is set to ‘Y’, then the sale is drop shipped and there is no update for stock on hand. Drop shipments are used for sales at a virtual or physical location where an order is taken from a customer, but the goods are shipped directly from the vendor to the customer (not via any store or warehouse owned by the retailer). If an item is used only for drop shipments and there is no stock on hand before or after the cost price is changed, the weighted average cost is never updated when average cost accounting method is used. The average cost will be the initial cost price at the time the item is set up. Over a period of time, under average cost accounting method, the cost price used to charge these items will drift away from the actual supplier cost. See SYSTEM_OPTIONS.STD_AV_IND for further details on cost accounting method.

If an off_retail amount was identified for the item/location, call the write_off_retail_markdowns() function to write tran_data records (tran_type 15) to record the difference. If the complex_promo_allowed_ind = ‘N’ and the item is on promotion, or if the complex_promo_allowed_ind = ‘Y’ and the TDETL total discount amount is greater than zero, write a promotional markdown. Note: this will also record a tran_data record (tran_type 15) for a TDETL record that has a promotional transaction type with no promotion number in order to record the markdown.

If an employee discount TDETL record has been encountered, a tran_data record with tran_code 60 will be written.

If the item is a wastage item, a tran_data record with tran_code 13 will be written. This record is used to balance the stock ledger, it accounts for the amount of the item that was wasted in processing.

process_detail_error()

This function writes a record to the load_err table for every non-fatal error that occurs.

set_counters()

Depending on the action passed into this function, it will either set a savepoint and store the values of counters or rollback a savepoint and reset the values of certain counters back to where they were originally set. This function is called when a non-fatal error occurs in the item_process() function to rollback and changes that may have been made.

calc_item_totals()

This function will set total retail and discount values including and excluding VAT, depending upon the store.vat_include_ind, system_options.vat_ind, complex_promo_allowed_ind, and the system_options.stkldgr_vat_incl_retl_ind.

calc_prom_totals()

This function will set promotional markdown values including and excluding VAT, depending upon the complex_promo_allowed_ind and the system_options.stkldgr_vat_incl_retl_ind. If the multi_prom_ind is on, the promotional markdown is the sum of the TDETL discount amounts. If the multi_prom_ind is off, the promotional markdown is the difference between the price_hist record with a tran_code of 0,4,8,11 and the price_hist record with a tran_code of 9 multiplied by the total sales quantity. Also, the tran_data old and new retail fields are only written if the multi_prom_ind is off.

Where vendor funding is present, compute the vendor contributions of the promotional discount in local and deal currencies, write local currency vendor funding invoices with tran_code = 6 to tran_data, and write deal currency vendor funding details to the deal_actuals_item_loc in deal currency. Call calc_vendor_funding (passing in the ex-vat total promotional mark down), to compute each vendor contribution (if any) in local currency for writing to the stock ledger and in deal currency for writing to deal_actuals_item_loc.

calc_vendor_funding()

This function accepts an ex-vat promotional discount amount and splits it by percentage for each of the vendors and deals in the list in both local and deal currency. A call is made to de-encapsulated currency conversion module convert(…), for efficiency in place of calling the PL/SQL equivalent function

process_sales_and_returns()

If a non-pack concession item is being processed, concession_data() is called to write accounts receivable data to the concession_data table. If the item is on consignment and not a packitem, the consignment_data() function will be called to perform consignment processing. The function write_tran_data() will be called to write a tran_data record with a tran_type 1 (always written), a tran_type 2 (if the system_options. vat_ind = Y and system_options.stkldgr_vat_incl_retl_ind = Y), a tran_type 3 (for non-inventory/non-deposit container item sales and returns), and a tran_type 4 (if the transaction was a return). If the transaction is a return, any tran_data records with tran_types of 1 and 2 will be written with negative retails. Also the update_price_hist() function will be called to update the most recent price_hist record.

If the retail price has changed since the sale occurred, process_reversal_records() function is called to write a tran_data record to reverse the price change for the items sold. Either a cancel markup or cancel markdown code is written. The retail amount to be cancelled is the difference between the retail sale price and current retail price multiplied by the total number of items sold or returned.

process_reversal_records()

If the retail price has changed since the sale occured, an unjustified loss on the stock ledger vs. the store tables is created. To correct this, a record needs to be written to tran_data reversing the price change for the items sold. This will use either a cancel markdown or markdown code. The quantity and retail will be the negative of the actual qty and retail, since a reversal is being processed.

validate_FHEAD()

Do standard string validations on input fields. This includes null padding fields, checking that numeric fields are all numeric, and validating the date field. If any errors arise out of these validation checks, return non-fatal error then set non-fatal error flag to true. This function will also validate the store location exists.

If the sales audit indicator is on currency and vat information will be provided in the file that has already been validated.

get_eow_eom_date()

This function returns the eow_date and eom_date for the current tran_date. For the eom_date, the appropriate base function is called to return the correct date for Gregorian or 454 calendar.

validate_THEAD()

Do standard string validations on input fields. This includes null padding fields, left shifting fields, checking that numeric fields are all numeric, placing decimal in all quantity and value fields, and validating the date field. If any errors arise out of these validation checks, return non-fatal error then set non-fatal error flag to true. This function will also validate the reference item exists.

If a reference item is passed in from the input file, retrieve the item for the reference item. Once the item is an item, retrieve the transaction and item level values, pack indicator, department, class, subclass, waste_type, waste_pct. Once this information is retrieved, check that the item/location relationship exists for the appropriate item type and call check_item_lock() and/or check_pack_lock depending on item type to lock this item's ITEM_LOC record.

If the sale audit indicator is 'Y' on system_options, the item will be a item and the dept, class, subclass, item level, transaction level and pack_ind will be included in the file. The UOM is assumed to already by have been converted to the standard UOM by Sales Audit.

If the Sales Audit indicator is 'N' on system_options, the UOM at which the item was sold will be compared with the items standard UOM value. If they are different, the quantity will be converted to the standard UOM amount. The ratio of the difference will also be computed and saved for use by validate_TDETL().

If an item is a wastage item set the wastage qty. The qty sent in the file shows the weight of the item sold. The wastage qty is the qty that was processed to come up with the qty sold. So if .99 of an item was sold, and item wastage percent is 10. The wastage qty is .99 / (1-.10) = 1.1 The wastage qty will be used through out the program except when writing tran_data records(see write_wastage_markdown) and daily_sales_discount records which will uses the processed qty from the file.

Class-level vat functionality is addressed here. The c_ get_class_vat cursor is fetched into the pi_vat_store_include_ind variable if vat is tracked at the class level in RMS (SYSTEM_OPTIONS.VAT_IND = 'Y' and SYSTEM_OPTIONS.CLASS_LEVEL_VAT_IND = 'Y'). The vat inclusion indicator passed in the input file is overwritten with the vat indicator for the class passed in the THEAD record of the input file.

If catchweight_ind is Y, call valid_all_numeric() to check that the actualweight_qty is all numeric, else call all_blank() to validate that it is blank. If the catchweight_ind is Y, convert actualweight_qty to 4 places of decimals reflecting the correct sign. Validate that the subtrans_type is either A, D or null.

If the item is part of an item transformation (pi_item_xform is TRUE), call get_item_xform_detail() to populate the pr_xform_items structure with the associated orderables, and return the total yield for all rows retrieved and also the calculated unit cost of the sellable item based on its component orderable items. This value overwrites pd_unit_cost_loc, which for standard items is populated by function item_check(…). If the returned sum of all retrieved pr_xform_items.as_yield does not equal 1, reject the record

get_ref_item()

This function is being called by the validate_THEAD function if the item_type is 'REF'. This function will return the item_parent of a specific item.

get_item_info()

This function gets item data from item_master and deps for an item_id passed in.

validate_TDETL()

This function will perform validation on the TDETL records passed into the program. The standard string validation on these fields includes null padding fields, left shifting fields, checking that numeric fields are all numeric, placing decimal in all quantity and value fields, and validating the date field. If any errors arise out of these validation checks, return non-fatal error then set non-fatal error flag to true.

The quantity is multiplied by the UOM ratio determined in validate_THEAD().

If a promotional transaction type is passed in, verify it is valid. If a promotional transaction type is passed in, but it is not valid, return non-fatal error then set non-fatal error flag to true.

If the item is a wastage item set the tdetl wastage qty. This is done the same way as setting the THEAD wastage qty.

If the promotion type is 9999 (i.e., all promotion types), verify that the promotion and promotion component are all numeric. If the promotion type is not 9999 (i.e., non-promotional), then verify that the promotion and promotion component are blank. If the promotion type is 9999, call validate_prom_info.

uom_convert()

This function is called by validate_THEAD to convert the selling UOM to the standard UOM.

validate_prom_info()

This function looks up the promotion in the rpm_promo table and the promotion_component in the rpm_promo_comp table. If either row does not exist, an error is reported and the function returns non-fatal. At the same time, any promotional consignment rate is retrieved and returned to the calling function

get_deal_contribs()

This function re-sizes the arrays to receive the list of vendor funding details if necessary and then appends the arrays with data, leaving a contribution count of zero or more in pl_deal_contribs_ctr. The function also fetches records from the deal_head, deal_comp_prom and deal_actuals_forecast tables to variables that will be used by the batch program in later processing. This function can process multiple promotions per deal component.

item_store_cursors()

This function checks the item_loc for the item / store combination. It is called by the item_check() and item_check_orderable().

new_item_loc()

This function creates a new store item relationship for items. It is called by item_check.

item_check()

This function verifies the fashion item/location relationship exists. It is only called when the item being processed is a fashion item. If the item/location relationship does not exist, it is created and a record is written to the Invalid item/location output file.

item_check_orderable()

This function gets the item information of a transform orderable item. If orderable pack indicator of the item is 'Y', call pack_check_orderable(). Else, it calls on the item_store_cursors function to check if location exists for the item. If none, it calls on procedure NEW_ITEM_LOC to create new store item relationship for the items.

pack_check_orderable()

This function calls on procedure NEW_ITEM_LOC to create new store item relationship for the items.

get_vat_rate()

This function calls on package VAT_SQL.GET_VAT_RATE and returns the vat rate of a specific item. This is being called by pack_check() and fill_packitem_array().

pack_check()

This function verifies the pack item/location relationship exists and retrieves the component items for the packitem. It is only called when the item being processed is a packitem. The component item, system indicator, department, class, subclass, cost, retail, price_hist retail, and component item quantity are fetched. If the packitem/location relationship does not exist, it is created for the Packitem and all of its components and a record is written to the Invalid item/location output file for the packitem.

The component items price ratios are also calculated. This indicates the retail contribution the component item gives towards the unit retail of the packitem. This ratio is calculated by taking the price_hist unit retail of the component divided by the total price_hist retail of all the component items for the packitem. Below is an example of how this ratio is calculated:

	Unit Retail	Qty	Retail	Calculation	Ratio
packitem A	$60				
item 1	$15	2	$30	($30/$90) * $60	.3333
item 2	$10	6	$60	($60/$90) * $60	.6667

item_supplier()

This function populates item information for the given item's supplier. This is called from the item_process() function, if the item_type is not = ‘PACK’ item.

get_unit_retail()

This function retrieves the current unit retail and the retail price of the item at the time of the sale from price_hist for the item/location being processed. If a tran_code of 8 is returned, the item is on clearance. The function will always return retail that are vat inclusive. If retail is stored in RMS with out vat (system_options.class_level_vat_ind = Y and class.class_vat_ind = Y) it will add vat to the retails.

get_base_price()

This function gets the unit_retail from price_hist (tran_type 0).

daily_sales_insert_update()

This function is called by write_off_retail, write_in_store, and process_daily_sales_discount. It performs the actual insert or fills a update array for the daily_sales_discount table.

check_daily_exists()

This function will check the daily_sales_discount for the existence of a record matching the input parameters.

process_daily_sales_discount()

This function will insert/update a record to daily_sales_discount for each TDETL record that has a promotional transaction type except employee discounts. Employee discount records are not written to daily_sales_discount, they are put on tran_data with a tran_code of 60. When employee discount records are encountered, values are set for the tran_data insert and the discount amount is added to the total sales value. This is done so employee discounts do figure into the promotional and in store calculations. When the multi_prom_ind is on all promotion types except employee discount will be ignored.

write_in_store()

This function will handle record sent in as ‘is store’ discounts amounts. It will call check_daily_exists and daily_sales_insert_update.

write_off_retail()

This function will calculate discrepancies between the amount sold for an item, and the amount it should have sold for (price_hist record). If these amounts are not in balance, a record is written to the daily_sales_discount table with a prom_type of ‘in store’ for reporting.

remove_stklgdr_vat()

This function will remove vat from 3 fields after the daily_sales_discount processing is complete. The variables od_off_retail_amt, od_new_retail, and od_old_retail are stripped of vat by calling vat_convert if the stock ledger does not contain vat.

write_off_retail_markdowns()

The write_tran_data() function will be called to write the off_retail markdown unless the item is on consignment or the off_retail amount is zero.

write_promotional_markdowns()

The write_tran_data() function will be called to write the promotional markdown unless the item multi_prom_ind is off and the transaction is a return, the item is on consignment, or the promotional markdown amount is zero. The tran_data new and old retails are only written if the multi_prom_ind is off. If any vendor funding rows are in the pr_deal_contribs arrays, call function write_vendor_tran_data to write the vat-inclusive vendor funding information to tran_data, and call function write_vendor_deal_actuals to write the vat-exclusive vendor funding information to deal_actuals_item_loc

write_vendor_tran_data()

This function writes a deal contribution record to the stock ledger for each of the vendor contributions stored in the deal contributions arrays by calling write_tran_data for the TRAN_CODE_VENDOR_FUNDING tran_type (type 6).

write_wastage_markdown()

This function will call to the write_tran_data() function if the item is a wastage item. A wastage item is an item that loses some of its weight (value) in processing. For example, a 1 pound chicken is broiled and loses 10% of its weight. The item is sold at .9 pounds, but in reality selling that .9 pounds of chicken removes 1 pound of chicken from the inventory. This function writes a tran_code 13 tran_data record to account for the amount of the chicken that was lost due to wastage in processing.

process_items()

Update the stock on hand on the item_loc_soh table for Sales and Returns unless the item is on consignment, drop shipped, non-inventory or concession. The SOH is updated for all the orderable components of a transformed item, but not the sellable component. Also, update the item_loc_hist table for Sale transactions. Do not update for returns.

Sales history is updated at week level and also, if the Gregorian calendar is in use (ps_cal_454_ind= 'N'), at month level. Additionally, sales history is updated for both sellable and orderable components of transformed items.

process_pack()

Update the stock on hand on the item_loc_soh table for Sales and Returns. Also, update the item_loc_hist table for Sale transactions (week-level sales history for pack items, and also month-level sales history if the Gregorian calendar is in use). Do not update for returns.

process_packitems()

This function performs processing for the component items of the packitems. This would include updates/inserts into stake_item_loc, edi_daily_sales, item_loc, item_loc_hist, vat_history_data, and tran_data. All of these tables do not write records at the packitem level, but at the component item level. When figuring retails to write to these tables, the component items price ratio should always be applied against the packitems retail to come up with the correct retail for each component item. If an employee discount TDETL record has been encountered, an tran_data record with tran_code 60 will be written for each component item.

write_tran_data()

Writes a record to the tran_data insert array.

write_edi_sales()

Writes a record to edi_daily_sales.

update_snapshot()

Calls the UPDATE_SNAPSHOT_SQL.EXECUTE function to update the stake_sku_loc and edi_daily_sales tables for late transactions.

get_454_info()

Calls on the CAL_TO_454 procedure to get the equivalent 454 info of a given date.

write_vat_err_message()

This function will create and write to the VAT output file when an item does not have VAT information setup when it is expected.

vat_history_data()

Writes a record to the vat_history table. History will only be written for the sellable item, not the orderable, and the orderable will never appear in the POS file.

consignment_data()

This function will perform processing for consignment items. Consignment items are such when the item_supplier table has a consignment rate applied to it. Consignment is when a retailer will allow a third party to operate under its umbrella and be paid for what it sells. An example of consignment may be a mass-merchant who consigns the magazine section of their store to a magazine vendor. The magazine vendor would have control over keeping the product stocked within the store. When a magazine is sold, the retailer would get paid for the magazine, then the retailer would essentially buy the magazine from the vendor. The consignment cost paid by the retailer to the vendor is the VAT-inclusive retail multiplied by the consignment rate divided by 100. So if the VAT-inclusive retail price of a magazine was $10 and the consignment rate was 50, the consignment cost would be $5.

Also a completed order to the vendor should be found/created for the supplier with an orig_ind = 4 (consignment). Consignment type invoices will be created for all PO’s created for consignments if the system_options.self_bill_ind is ‘Y’.

Purchase order headers are created at supplier, supplier/dept, supplier/dept/location or supplier/dept/location/item levels depending on the system_options flag gen_con_invc_itm_sup_loc being S, L or I. Purchase orders are matched 1 to 1 with sales invoices, but for returns there is no purchase order and an invoice is created for every transaction regardless of the consolidation level. The flag system_options.gen_con_inv_freq can have values P (multiPle), W (Weekly), M (Monthly), or D (Daily). This controls the date used for the 1 to 1 matching which is vdate, eow_date, eom_date or vdate respectively.

Also a tran_data record (tran_type 20) will be written to record the consignment transaction to the stock ledger. The retails should be VAT inclusive or exclusive, depending on the system_options.stkldgr_vat_incl_retl_ind.

This function uses support functions: check_order(), order_head(), invc_data(), to handle the order creation-update and the invoice creation-update.

If a promotional consignment rate is present for the current promotion, over-write that returned from item_supplier

order_head()

This function inserts records into ordhead to create new orders (except for return consignment items). It sets the location to the current store number if the gen_con_invc_itm_sup_loc_ind flag is I or L, otherwise (for S) should set null. The order date is set depending on system_options.gen_con_inv_freq. The values are P (multiPle), W (Weekly), M (Monthly), or D (Daily). This controls the date used for the 1 to 1 matching which is vdate, eow_date, eom_date, or vdate respectively.

invc_data()

This function inserts/updates invc_head, invc_detail records if invc_match ind is 'Y'. Before writing the invoice records, the retail and consignment cost are converted to the associated order's currency.

The system_options parameter system_options.gen_con_invc_itm_sup_loc_ind carries values S, L or I and states the level at which separate invoices are to be generated for each supplier/dept(S), supplier/dept/location(L) or item/supplier/location(I). When a new invoice at the appropriate level is created, then for gen_con_invc_itm_sup_loc_ind values L and I, an invc_xref row is also created to link the invoice to the target location

find_and_fill_invc_detail ()

This function fills the invc_detail, updates the array and posts if the array is full

get_prom_type_info()

This function will retrieve all valid promotional transaction types from the code_detail table. Valid promotional transaction types are those where the code_type = ‘PRMT’.

get_uom_classes()

This function loads all the uom codes and their classes into a global table for look up during THEAD processing.

get_item_xform_details()

This function populates the pr_xform_items structure with the associated orderables, and returns the total yield for all rows retrieved, and also the calculated unit cost of the sellable item based on its component orderable items. This value overwrites pd_unit_cost_loc, which for standard items is populated by function item_check(…). If the returned sum of all retrieved pr_xform_items.as_yield does not equal 1, reject the record.

The processing to do this is de-encapsulated from packaged function ITEM_ XFORM_ SQL.CALCULATE_COST, as this is expected to be more efficient than calling the packaged function directly. The de-encapsulated logic is performed by the following three functions: get_loc_item_retail(), get_orderable_cost(), get_orderable_retail().

get_loc_item_retail()

This function returns the unit_retail from item_loc. If a unit retail for the input item/location combination does not exist on the item_loc table, a call is made to retrieve the unit retail from RPM (via the PRICING_ATTRIB_SQL.GET_BASE_ZONE_RETAIL package function).

get_orderable_cost()

This function returns unit_cost from item_supp_country_loc or item_supp_country.

get_orderable_retail()

This function returns the unit_retail for each sellable item, computes the apportioned sellable retail and adds it into the returned total orderable retail.

fill_packitem_array()

This function will retrieve the component items for a packitem with the appropriate item level information into an array.

write_item_store_report()

This function will create and write to the Invalid item/location output file when an item does not exist at a location it was sold/returned at.

posting_and_restart()

Post all array records to their respective tables and call restart_file_commit to perform a commit the records to the database and restart_file_write to append temporary files to output files.

post_tran_data()

This function inserts records in the tran_data table. This is called by posting_and_restart function.

post_item_loc()

This function updates the stock_on_hand of the item_loc_soh table. This is called by posting_and_restart function.

post_item_loc_hist()

This function updates the various fields (sales_issues, value, gp, last_update_datetime and last_update_id) of the item_loc_hist table. This is called by posting_and_restart function.

post_item_loc_hist_mth()

This function updates the various fields (sales_issues, value, gp, last_update_datetime and last_update_id) of the item_loc_hist_mth table. This is called by posting_and_restart function.

post_pack()

This function updates the various fields (last_hist_export_date, first_sold, last_sold, qty_soldm, last_update_datetime and last_update_id) of the item_loc_soh table. This is called by posting_and_restart function.

post_packstore_hist()

This function updates the various fields (sales_issues, value, retail, last_update_datetime and last_update_id) of the item_loc_hist table. This is called by posting_and_restart function

post_packstore_hist()

This function updates the various fields (sales_issues, value and retail) of the item_loc_hist_mth table. This is called by posting_and_restart function.

post_vat_hist_upd()

This function updates the various fields (vat_amt, last_update_datetime and last_update_id) of the vat_history table. This is called by posting_and_restart function.

post_ edi_daily_sales_upd ()

This function updates sales_qty of the edi_daily_sales table. This is called by posting_and_restart function.

post_daily_sales_discount ()

This function updates the various fields (sales_qty, sales_retail, discount_amt, expected_retail and actual_retail) of the daily_sales_discount table. This is called by posting_and_restart function.

post_invc_detail_upd ()

This function inserts into the invc_detail_temp table. This is called by posting_and_restart function.

post_invc_detail_upd ()

This function inserts into invc_head_temp table. This is called by posting_and_restart function.

size_arrays()

This function allocates memory for the arrays used in this program.

resize_arrays()

This function reallocates memory for the insert arrays.

write_lock_rej()

This function will write the current record set from the input file (THEAD-{TDETL}-TTAIL) that was rejected due to lock error to the lock file.

concession_data()

This function inserts records into concession_data for non-pack concession items.

deal_actuals_insert_update ()

This function accepts a list of primary key values and update values for the deal_actuals_item_loc table, and a row_id which is null if the row does not exist yet. If it does not exist, a new row is inserted, otherwise the row_id and update values are written to the holding array, for bulk update later.

check_deal_actuals_exists()

This function accepts a list of primary keys for table deal_actuals_item_loc, does a look up and returns the row_id or null if it exists, or not.

write_vendor_deal_actuals ()

This function causes actual vendor contribution amounts to be written to the deal_actuals_item_loc table for each of the computed vendor funding contributions held in the pr_deal_contribs array. Calls check_deal_actuals_exists to check if each target primary key set exists, and calls deal_actuals_insert_update to insert a new row, or write update information to the holding array if a row already exists.

post_deal_actuals ()

This function updates the various fields (actual_turnover_units, actual_turnover_revenue and actual_income) of the deal_actuals_item_loc. This is called by posting_and_restart function.

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 function by passing pointer to detail record structure, number of bytes in structure, and reject file pointer, or use the write_lock_rej() function to write to the lock reject file in case the non-fatal error was a lock error,

Input File:

The input file should be accepted as a runtime parameter at the command line. All number fields with the number(x,4) format assume 4 implied decimal included in the total length of 'x'.

When the system_options field sa_ind is 'Y' the following FHEAD fields will be populated and already validated: Vat include indicator, Vat region, Currency code, and Currency retail decimals. When the sa_ind is 'N' these values will not be used and retrieved from the system.

When the system_options field sa_ind is 'Y' the following FHEAD fields will be populated and already validated: Item Level, Transaction Level, Pack_ind, Dept, Class, and Subclass. When the sa_ind is 'N' these values will not be used and retrieved from the system. Also, the UOM at which the item was sold will been converted to the standard UOM for the item. When the sa_ind is on, all items are assumed to be items.

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	File Type Definition	Char(4)	POSU	Identifies file as 'POS Upload'
	File Create Date	Char(14)	create date	date file was written by external system
	Location Number	Number(10)	specified by external system	Store identifier
	Vat include indicator	Char(1)		Determines whether or not the store stores values including vat. Not required but populated by Retek sales audit
	Vat region	Number(4)		Vat region the given location is in. Not required but populated by Retek sales audit

Record Name	Field Name	Field Type	Default Value	Description
	Currency code	Char(3)		Currency of the given location. Not required but populated by Retek sales audit
	Currency retail decimals	Number(1)		Number of decimals supported by given currency for retails. Not required but populated by Retek sales audit
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies transaction record type
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	Transaction Date	Char(14)	transaction date	date sale/return transaction was processed at the POS
	Item Type	Char(3)	REF ITM	item type will be represented as a REF or ITM
	Item Value	Char(25)	item identifier	the id number of an ITM or REF
	Dept	Number(4)	Item’s dept	Dept of item sold or returned. Not required but populated by Retek sales audit
	Class	Number(4)	Item’s class	Class of item sold or returned. Not required but populated by Retek sales audit
	Subclass	Number(4)	Item’s subclass	Subclass of item sold or returned. Not required but populated by Retek sales audit

Record Name	Field Name	Field Type	Default Value	Description
	Pack Indicator	Char(1)	Item's pack indicator	Pack indicator of item sold or returned. Not required but populated by Retek sales audit
	Item level	Number(1)	Item's item level	Item level of item sold or returned. Not required but populated by Retek sales audit
	Tran level	Number(1)	Item's tran level	Tran level of item sold or returned. Not required but populated by Retek sales audit
	Wastage Type	Char(6)	Item's wastage type	Wastage type of item sold or returned. Not required but populated by Retek sales audit
	Wastage Percent	Number(12)	Item's wastage percent	Wastage percent of item sold or returned. Not required but populated by Retek sales audit
	Transaction Type	Char(1)	'S' – sales 'R' - return	Transaction type code to specify whether transaction is a sale or a return
	Drop Shipment Indicator	Char(1)	'Y' 'N'	Indicates whether the transaction is a drop shipment or not. If it is a drop shipment, indicator will be 'Y'. This field is not required, but will be defaulted to 'N' if blank.

Record Name	Field Name	Field Type	Default Value	Description
	Total Sales Quantity	Number(12)		Number of units sold at a particular location with 4 implied decimal places.
	Selling UOM	Char(4)		UOM at which this item was sold.
	Sales Sign	Char(1)	‘P’ - positive ‘N’ - negative	Determines if the Total Sales Quantity and Total Sales Value are positive or negative.
	Total Sales Value	Number(20)		Sales value, net sales value of goods sold/returned with 4 implied decimal places.
	Last Modified Date	Char(14)		For VBO future use
	Catchweight Indicator	Char(1)	NULL	Indicates if item is a catchweight item. Valid values are ‘Y’ or NULL
	Actual Weight Quantity	Number(12)	NULL	The actual weight of the item, only populated if catchweight_ind = ‘Y’
	Sub Trantype Indicator	Char(1)	NULL	Tran type for ReSA Valid values are ‘A’, ‘D’, NULL
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies transaction record type

Record Name	Field Name	Field Type	Default Value	Description
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	Promotional Tran Type	Char(6)	promotion type – valid values see code_detail table.	code for promotional type from code_detail, code_type = 'PRMT'
	Promotion Number	Number(10)	promotion number	promotion number from the RMS
	Sales Quantity	Number(12)		number of units sold in this prom type with 4 implied decimal places.
	Sales Value	Number(20)		value of units sold in this prom type with 4 implied decimal places.
	Discount Value	Number(20)		Value of discount given in this prom type with 4 implied decimal places.
	Promotion Component	Number(10)	NULL	Links the promotion to additional pricing attributes
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	Transaction Count	Number(6)	specified by external system	Number of TDETL records in this transaction set

Record Name	Field Name	Field Type	Default Value	Description
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	specified by external system	ID of current line being processed by input file.
	File Record Counter	Number(10)		Number of records/transactions processed in current file (only records between head & tail)

Invalid Item/Store File:

The Invalid Item/Store File will only be written when a transaction holds an item that does not exist at the processed location. In the event this happens, the relationship will be created during the program execution and processing will continue with the item and store number being written to this file for reporting.

VAT File:

The VAT file will only be written if a particular item cannot retrieve a VAT rate when one is expected (e.g. the system_options.vat_ind is on). In this event, a non-fatal error will occur against the transaction and a record will be written to this file and the Reject file.

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

Assumption: Variable weight UPCs are expected to already be converted to a VPLU with the appropriate quantity.

Output Specifications

N/A

Scheduling Considerations

Processing Cycle:	PHASE 2 (daily)
Scheduling Diagram:	This program will likely be run at the beginning of the batch run during the POS polling cycle. It can be scheduled to run multiple times throughout the day, as 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 sales/returns upload module will be a valid item sales transaction at a given store location. The location type will be inferred as a store type and the item can be passed as an item or reference item type. The logical unit of work will be defined as a number of these transaction records. The commit_max_ctr field on the restart_control table will determine the number of transactions that equal a logical unit of work.

The file records will be read in groups of numbers equal to the commit_max_ctr. After all records in a given read are processed (or rejected either as a reject record or a lock error record), the restart commit logic and restart file writing logic will be called, and then the next 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 10000 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 or written to the lock file depending on the reject reason. If a fatal error is returned, then 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.

Sales audit export to GL [saexpgl] batch design

Design Overview

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

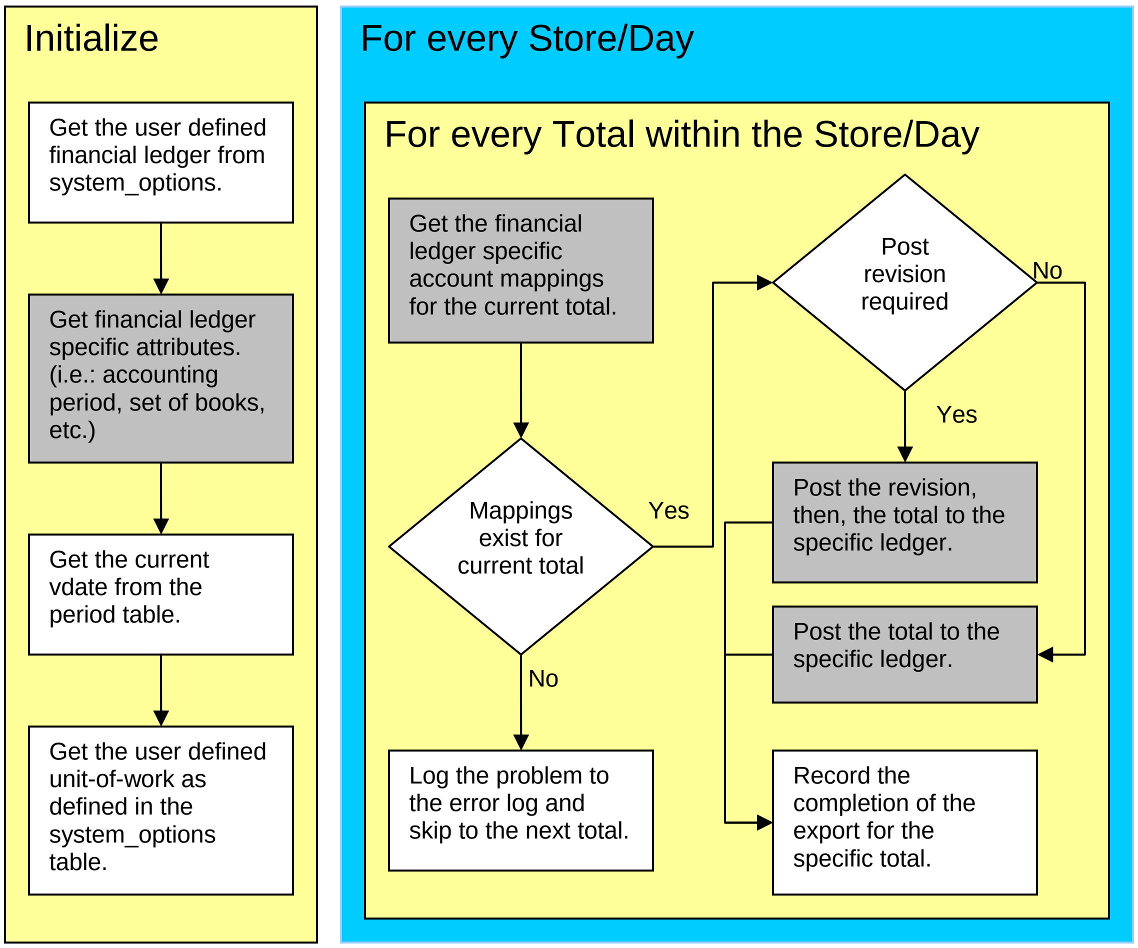
This version of the program is meant for the interface between RMS 11.0 and Oracle Financials.

Tables Affected:

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

Program Flow

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


Initialize
Get the user defined financial ledger from system_options.
Get financial ledger specific attributes. (i.e.: accounting period, set of books, etc.)
Get the current vdate from the period table.
Get the user defined unit-of-work as defined in the system_options table.
For every Store/Day
For every Total within the Store/Day
Get the financial ledger specific account mappings for the current total.
Mappings exist for current total
Yes
No
Log the problem to the error log and skip to the next total.
Post revision required
No
Yes
Post the revision, then, the total to the specific ledger.
Post the total to the specific ledger.
Record the completion of the export for the specific total.

Legend:

▨ Process specific to user defined financial application.

☐ Process generic to all financial applications.

Global Variable Descriptions

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

Function Level Description

main()

Check command line for required arguments.

Call LOGON to connect to the database.

Call Init to initialize the program.

Call process to export the available RMS data.

Report unlocking errors.

Call final to cleanup.

init()

Call retek_init.

Get the current vdate from the period table, using fetchVdate.

Get the user financial application type from system_options.financial_app.

‘O’ = Oracle GL

‘P’ = PeopleSoft GL

Get the Financial application specific attributes (i.e. accounting period information, set of books identntifier, etc.)

If Oracle GL, retrieve the following details as defined in the RMS database:

Fif_gl_setup.set_of_books_id

Fif_gl_setup.last_update_id

Get and save the value of sa_system_options.unit_of_work, by calling the function fetchSaSystemOptions.

process()

Retrieve a store/day by calling fetchStoreDayToBeExported.

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

Find out the number of errors pending for the store/day by calling fetchStoreDayErrorCount.

If the unit of work is store and the number of errors in the store/day is greater than zero, then release the lock by calling release_lock and skip the store/day, otherwise continue.

Retrieve a total to export by calling getTotal.

If Oracle GL, check to ensure that the selected total has a user defined cross-reference in the sa_fif_ora_cross_ref table by calling the function getOracleMapping. If a mapping (Oracle CCID) does not exist for the selected total log the problem in the Retek error log and go onto the next total.

If the tran_sign is 'N' (code_type is SAFD), the currenct retrieved value will be post to Oracle with negative sign.

Post the current total to the GL by calling the financial application specific function:

If Oracle, call postOracleGL

If there are more totals for the selected store/day, loop through the store day totals (getTotal).

Call the library function markStoreDayExported.

Call release_lock and go on to the next store/day.

ProcessStoreDay()

Get all the totals for the store/day by calling getTotal().

For each Total_id, call getOracleMapping() for Oracle account.

If Status returned from getTotal() is 'N'. The opposite amounts will be posted to the Stg_fif_gl_data table (that is, send a negative number).

Call UpdateGLArray() to populate gl_data_array for inserting stg_fif_gl_data table.

Call the library function markTotalExported and include the current period number. This function has to be called once for each total that is exported.

CanProcess()

Calling fetchStoreDayErrorCount to find out the number of errors pending for the store/day.

If the unit of work is store and the number of errors in the store/day is greater than zero, skip the store/day and write to the if_errors for the store/day.

final()

Clean up – free any memory used.

Call retek_close.

AddToList()

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

DeleteList()

This function deletes linked list, and free the memory.

GetNext()

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

RemoveFromList()

This function removes processed store/day from linked list.

SizeGlDataArray()

This function allocates memory for gl_data_array.

ProcessLockedSD()

This function locks the store/day to be processed.

GetOracleMapping()

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

InsertToOracleGL()

This function inserts the record processed into STG_FIF_GL_DATA table.

UpdateGLArray()

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

STG_FIF_GL_DATA column explanation

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

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

WriteErrorTable()

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

Stored Procedures / Shared Modules (Maintainability)

Shared Module	Module Description
libresa.a	ReSA Library
get_lock	used to establish a read lock on a store/day
release_lock	used to release a store/day lock
fetchStoreDayToBeExported	This fetches all store days that are ready for export for a given usage type.
getTotal	This fetches all totals that can be exported for the given usage type and for the given store day.
fetchStoreDayErrorCount	This functions returns the number of errors pending for a given store day.
markTotalExported	records the passed total as exported
markStoreDayExported	records the passed store day as exported
fetchSaSystemOptions	This function retrieves all entries in the sa_system_options table.
fetchVdate	This function retrieves the vdate from the period table.

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

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

Input/Output Specifications

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

Integrity Constraints

Processing Cycle:	Anytime – Sales Audit 11.0 is a 24/7 system.
Scheduling Diagram:	This program will be run after the ReSA totaling process: satotals.pc and sarules.pc.
Threading Scheme:	N/A

Restart / Recovery

The logical unit of work for this module is defined as a unique store/day combination. Records will be fetched, updated and inserted in batches of pi_commit_max_ctr. Only one commit will be done: at the end, after a store/day has been completely processed, a call to release_lock() performs a commit.

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

Driving cursor 1:

This driving cursor is embedded in the library function fetchStoreDayToBeExported(). Given a system code, of 'SYSE', this function fetches all store/days with a store_status of 'C'lose, a data_status of 'F'ully loaded and an audit_status of 'A'udited, 'S'tore errors pending or 'H'Q errors pending that are ready to export to the given system.

Driving cursor 2:

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