Retek® Merchandising System 10.1

Operations Guide Addendum

Retek

Corporate Headquarters:

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

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

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 e-mail, 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.

Contact Method	**Contact Information**
Internet (ROCS)	www.retek.com/support Retek's secure client Web site to update and view issues
E-mail	support@retek.com
Phone	US & Canada: 1-800-61-RETEK (1-800-617-3835) World: +1 612-587-5800 EMEA: 011 44 1223 703 444 Asia Pacific: 61 425 792 927
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

Introduction

This addendum to the RMS 10 Operations Guide presents changes that have resulted from work completed during RMS 10.1 development. RMS 10 Operations Guide volumes impacted include:

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

There are no changes to Volume 3, Batch Program Overview.

The batch schedule diagram accompanies this addendum. See the filename: rms-101-batchschedule.pdf. The only change to the batch schedule document is a correction to the title, which now states "Retek Merchandising System 10.1 Batch Schedule." No part of the schedule itself is changed from RMS 10.0.

This addendum contains one chapter for each volume. Changes for each volume are listed either by a note of the changes or by the inclusion of the entire changed document.

Volume 1 – Functional overviews

Price and POS download

For RMS 10.1, the following data are written to the POS_MODS table:

- When downloading item information to the POS system, the new unit retail includes the VAT indicator for the class, VAT code, and VAT rate for the item.
- Whenever a new item-location record is written to POS_MODS, VAT_CODE, VAT_RATE, and CLASS_VAT_IND are now included. See the overview “VAT maintenance” for more information. Also see the PL/SQL package POSUPDS.

EDI

The EDIUPCAT batch module now accepts up to four supplier differentiators when uploading items from a supplier.

Stock counts

The STKVAR batch module (Stock Count on Hand Updates) now performs the following task:

Checks the system VAT indicator, the indicator for stock ledger VAT, the class level VAT indicator, and the indicator for retail inclusion of VAT indicator for the class to determine if VAT needs to be added on, stripped off, or neither before updating the STAKE_PROD_LOC table. See the “VAT maintenance” overview for more information.

Differentiators

Differentiators have experienced a number of changes in RMS 10.1, including:

- Four diffs can now be associated with an item. In RMS 10.0, only two diffs could associate with an item.
- RMS can upload four (4) diffs from a supplier via the EDIUPCAT program.
- A user may not create more than 30 diff types.
- Diff types are now held on the new table DIFF_TYPE and are no longer held on the CODE_HEAD and CODE_DETAIL tables.
- The diff type maximum field length is six (6) characters, and the description field is 40 characters.
- A user may only delete a diff type if no diff groups or diff IDs are associated with that diff type.
- Whenever RMS publishes item messages to the RIB, it can include all four diffs and their types. See the “Items” functional overview in the operations guide for more information about RMS item message publication.

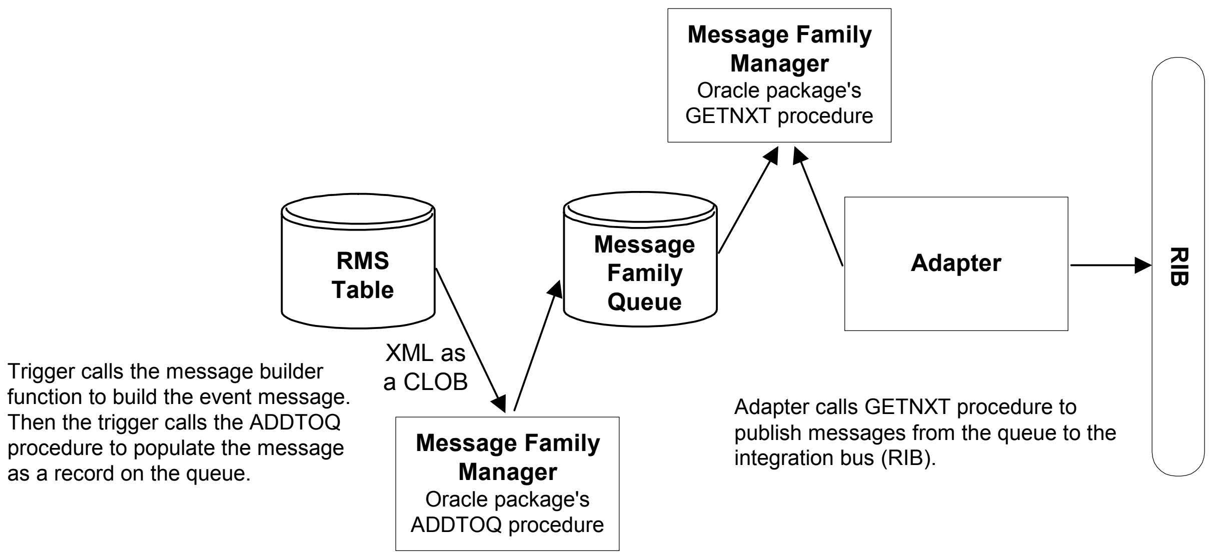
Promotions (prices)

The description for the program PCOVRLQ.PC is inaccurate in the RMS 10.0 Operations Guide. The following description replaces it:

PCOVRLQ.PC (Promotion Price Overlap)–This module locates and writes overlapping promotions to the PRICE_OVERLAP_LOG table for reporting. It also updates the promotion status to ‘Submitted’ or ‘Approved’ when no overlap records are found.

Message publishing

The publication of a message to the Retek Integration Bus is handled by the table trigger, the message family manager, and RMS's eWay (adapter). See the diagram and the explanations that follow. For RMS 10.1 changes, see the following page.


Message Family Manager
Oracle package's GETNXT procedure
RMS Table
Message Family Queue
Adapter
RIB
XML as a CLOB
Trigger calls the message builder function to build the event message. Then the trigger calls the ADDTOQ procedure to populate the message as a record on the queue.
Message Family Manager
Oracle package's ADDTOQ procedure
Adapter calls GETNXT procedure to publish messages from the queue to the integration bus (RIB).

1 Messages are specific to a 'family'. For example, the "Supplier" family includes all suppliers (vendors) and their addressses.

2 Messages are queued as records to a staging table called a Message Family Queue ([message family name]_MFQUEUE).

3 Each message family has a Message Family Manager that is a PL/SQL package. Two public procedures in the package are ADDTOQ and GETNXT.

4 An event on an RMS table (that is, an insert, update or delete) 'triggers' a record to be created in the message family queue. The trigger calls the message building procedure (for example, WORKORDER_XML.BUILD_MESSAGE) to build the XML as a CLOB.

5 The trigger calls the ADDTOQ procedure that generates sequence numbers and inserts the event message as a record on the queue.

6 The RIB adapter calls the GETNXT procedure to publish the message from the message family queue to the integration bus.

RMS 10.1 changes

In RMS 10.0, the BUILD_MESSAGE procedure calls the RIB_XML procedure to create the CLOB. RMS 10.1 retains this process but applies a new process called RIB_SXW for the following messages:

- ATP (available to promise)
- Items
- Purchase orders
- Stock orders (allocations and transfers)
- Work orders

RIB_SXW concatenates the XML strings in the CLOB. Using RIB_SXW in this way speeds the creation of CLOB building for these high volume messages.

Currency exchange rates

Currency exchange rates constitute financial information that is published to the Retek Integration Bus (RIB). A currency exchange rate is the price of one country's currency expressed in another country's currency. RMS 10.1 subscribes to a currency exchange rate message that is held on the RIB. After confirming the validity of the records enclosed within the message, RMS 10.1 updates its tables with the information.

This overview describes the following:

- A description of the specific data within the currency exchange rates terms message.
- A summary of the steps that occur during the processing of the currency exchange rates message.
- A summary of the currency exchange rates message that shows its corresponding DTD and mapping document.
- The tables in RMS 10.1 that are affected by the currency exchange rates subscription message.

Note: When the systems are initially set up, identical currency information (3-letter codes, exchange rate values) are entered into both the RMS and the financial system. If a new currency needs to be used, it must be entered into both the financial system and RMS before a rate change is possible. No functionality currently exists to bridge this data.

Currency exchange rates message

RMS 10.1 subscribes to a flat currency exchange rates terms message that consists of a currency exchange rate record. A currency exchange rates record can only be created or updated. Previous currency exchange rates records are not deleted.

RMS subscribes to a currency exchange rates message named CurrRateDesc.

Data in the currency rates subscription message that has primary significance to RMS includes:

- The exchange rate for the specified currency/type/effective date combination.
- The date on which the currency rate became or will become active.
- The type of exchange rate the history exists for. The values include:
  - C (Consolidation)
  - O (Operational)
  - L (Letter of Credit/Bank)
  - P (Purchase Order)
  - U (Customs Entry)
  - G (Logistics)

Currency exchange rates message subscription

The currency exchange rates message subscription process primarily consists of the two following PL/SQL packages:

RMSSUB_CURRATECRE and its CONSUME procedure serve as RMS's subscription API. A second package, RMSSUB_CURRXRATE, holds functions that performs the following tasks on data contained in the message:

- Parse
- Process
- Insert
- Update
- Validate

Currency exchange rates message subscription process

From a high-level perspective, the currency exchange rates message subscription process primarily consists of the following steps:

1 The RMS external currency exchange rates adapter (eWay) recognizes that a message with a currency exchange rates name exists on the RIB. The RIB calls the first package, which serves as RMS' subscription API: RMSSUB_CURRATECRE.CONSUME. This package initially processes the message and the XML CLOB contained in the message.

2 The package, RMSSUB_CURRXRATE, accepts an XML file in the form of an Oracle CLOB data type from the RIB. The procedure validates the XML file format. If the validation is successful, the XML in the message is parsed into PL/SQL. Note that in addition to calling other functions, the package calls CONVERT_TYPE, which converts FIF_EXCHANGE_TYPE to RMS_EXCH_TYPE through the table FIF_CURRENCY_XREF. The validated currency exchange rates data is inserted into the CURRENCY_RATES table.

Currency exchange rates message summary

The following table shows you the CurRateCre message, the document type definition (DTD) that describes the XML message, and the mapping document that describes the data contained in the message. The mapping document contains information that includes the source table, column, and data types. See the Retek 10.1 Integration Guide for more information regarding these documents.

Message Name	Type (DTD)	Mapping Document
CurRateCre	CurrRateDesc.dtd	Map_CurrRateDesc.xls

Currency exchange rates tables

The following description is for the primary tables in RMS 10.1 that hold currency exchange rates data and are used in message subscription processing:

CURRENCY_RATES

This table contains the exchange rates for every currency used in the system. A currency can have multiple exchange rates based on a combination of different exchange types and effective dates.

FIF_CURRENCY_XREF

This cross-reference table is used to translate the financial package's exchange type to the Retek-defined exchange type. During message processing, the CONVERT_TYPE function resolves the financial package's type by referencing the FIF_CURRENCY_XREF table, which translates the financial type into the RMS type. The RMS type is then written to the CURRENCY_RATES table. The FIF_CURRENCY_XREF table is populated at installation and is maintained by system administration. The table cannot be updated by other users.

Freight terms

Freight terms are supplier-related financial arrangement information that is published to the Retek Integration Bus (RIB), along with the supplier and the supplier address, from the financial system. Freight terms are the terms for shipping (for example, the freight terms could be a certain percentage of the total cost; they could be free; and so on). RMS 10.1 subscribes to a freight terms message held on the RIB. After confirming the validity of the records enclosed within the message, RMS 10.1 updates its tables with the information.

This overview describes the following:

- A description of the specific data within the freight terms message.
- A summary of the steps that occur during the processing of the freight terms message.
- A summary of the freight terms message that shows its corresponding DTD and mapping document.
- The table in RMS 10.1 that is affected by the freight terms subscription message.

Freight terms message

RMS 10.1 subscribes to a flat freight terms message that consists of a freight terms record. A freight term record can only be created or updated. Previous freight terms records are neither deleted nor modified; they are rendered enabled/disabled through a flag associated with the active/inactive date.

RMS 10.1 subscribes to the FrtTermDesc freight terms message .

Data in the freight terms message that has primary significance to RMS 10.1 includes:

- The number that uniquely identifies the freight terms.
- A description of the freight terms used in the system.
- The date for assigning an active date to the freight terms.
- The date for assigning an inactive date to the freight terms.

Freight terms message subscription

The freight terms message subscription process primarily consists of the three following PL/SQL packages:

RMSSUB_FRTTERMCRE and its CONSUME procedure serve as RMS' subscription API.

A second package, RMSSUB_FTERM, holds functions that perform the following tasks on data contained in the message:

- Parse
- Process

A third package internal to RMS, FTERM_SQL, holds functions that perform the following tasks on data contained in the message:

- Insert
- Update
- Validate

Freight terms message subscription process

From a high-level perspective, the freight terms message subscription process primarily consists of the following steps:

1. The RMS 10.1 external freight terms adapter (eWay) recognizes that a message with a freight terms name exists on the RIB. The RIB calls the first package, which serves as RMS' subscription API: RMSSUB_FRTTERMCRE.CONSUME. This package initially processes the message and the XML CLOB contained in the message.
2. The second package, RMSSUB_FTERM, accepts an XML file in the form of an Oracle CLOB data type from the RIB. The procedure validates the XML file format, and if the validation is successful, parses the XML in the message into PL/SQL.
3. The third package, RMSSUB_FTERM makes a call to RMS' FTERM_SQL, which performs the insert/update to the FREIGHT_TERMS table within RMS 10.1.

Freight terms message summary

The following table shows you the FrtTermCre message, the document type definition (DTD) that describes the XML message, and the mapping document that describes the data contained in the message. The mapping document contains information that includes the source table, column, and data types. See the Retek 10.1 Integration Guide for more information regarding these documents.

Message Name	Type (DTD)	Mapping Document
FrtTermCre	FreightTermDesc.dtd	Map_FreightTermDesc.xls

Freight terms table

The following description is for the primary table in RMS 10.1 that holds freight terms data:

FREIGHT_TERMS

This table contains one row for each set of freight terms allowed in the company. The table is populated during installation.

General ledger chart of accounts

Before RMS can publish stock ledger data to an external financial application, it must receive that application's general ledger chart of accounts (GLCOA) structure. RMS accomplishes this through a subscription process described in this overview.

A chart of account is essentially the financial application's debit and credit account segments (for example, company, cost center, account, and so on) that apply to the RMS product hierarchy. In some financial applications, this is known as CCIDs (Code Combination IDs). Upon receipt of GLCOA message data, RMS populates the data to the FIF_GL_ACCT table. The GL Cross Reference (glcross.fmb) form is then used to associate the appropriate department, class, subclass, and location financial data to a chart that allows the population of that data to the GL_FIF_CROSS_REF table.

Message summary

The following table lists the GLCOA message by its message type, the document type definition (DTD) that describes the XML message, and the mapping document that describes the data contained in the message. Consult the Retek 10.1 Integration Guide to view these documents.

Message Type	Type (DTD)	Mapping Document
GLCOACre	GLCOADesc.dtd	Map_GLCOADesc.xls

GLCOA message subscription

The GLCOA message subscription process consists of the following PL/SQL packages:

- RMSSUB_GLCOACRE and its CONSUME procedure serves as RMS' subscription API.
- A second package, RMSSUB_GLCACCT, holds functions that performs the following tasks on data contained in the message:
  - Parse the message header
  - Process the message header
- PROCESS_GLACCT – Accepts the input GL record and places the data into a local GL record, used in the package to manipulate the data. It calls the following support functions to perform all business logic on the record:
  - insert
  - update
  - validate

Data is populated on the FIF_GL_ACCT table.

System option for financial application

RMS’ SYSTEM_OPTIONS table holds the column FINANCIAL_AP, where the interface financial application is indicated. Settings in this column are either “O” or null. “O” indicates an external financial application. A null indicates that no financial application is interfaced with RMS.

Cost changes

Cost values serve as a starting point in the creation of a purchase order. RMS 10.0 introduces the multi-channel concept where stores and warehouses can be ‘virtual’ as well as physical locations. If RMS is set up to run multi-channel (meaning the multi-channel indicator on the SYSTEM_OPTIONS table is set to “Y” (yes)), only virtual locations hold stock. Physical warehouses, while not being stockholding locations, do hold supplier item cost that is shared across all virtual warehouses associated with the physical warehouse. This section describes how supplier cost changes are processed in RMS, with a focus on the batch modules SCCEXT and CCPRG.

Cost change process

The cost change process begins with the supplier form SUPPSKU. Changes made on this form impact these tables:

- COST_SUSP_SUP_HEAD, always populated.
- COST_SUSP_SUP_DETAIL, populated if the cost at the country level is changed. Otherwise COST_SUSP_SUP_DETAIL_LOC is populated if cost is being maintained at individual locations. Bracket cost data are also stored on these two tables.

If cost changes are updated directly from the supplier, the batch module EDIUPCAT indirectly populates the cost tables, using the following process: EDIUPCAT populates EDI_COST_CHG and EDI_COST_LOC. The RMS user can then accept EDI cost changes through the EDI cost change dialog. Accepted changes then populate the cost tables.

After updates to the cost tables occur, they are processed into the following tables by the SCCEXT module:

- ITEM_SUP_COUNTRY for the country level cost change. The program distributes the cost to all locations on ITEM_SUP_COUNTRY_LOC (for the current unit cost). Note that this table is always updated, regardless of the multi-channel indicator
- ITEM_SUPP_COUNTRY_BRACKET_COST if the supplier is bracket costing
- ITEM_LOC_SOH for locations

Multi-channel supplier cost change rules:

- Average cost is held on the ITEM_LOC_SOH table
- Cost changes are managed and stored at the physical warehouse level since the unit cost must remain consistent across all virtual warehouses within the same physical warehouse
- On the ITEM_LOC_SOH table, cost is held at the virtual level, to include physical stores

- A purchase order P.O. cannot be created for non-stockholding locations, like physical warehouses, and non-stockholding stores, like Web stores and catalog stores
- Each physical and virtual store has a default virtual warehouse
- Cost changes sent by a supplier and uploaded by the batch module EDIUPCAT apply to the physical warehouse before quantities are apportioned to the virtual warehouses in SCCEXT.PC
- When cost changes are received from a supplier via EDI, two outcomes are possible for updating the system costs. If the item is in a Worksheet or Submitted status, system costs are updated online when the cost change is accepted in the EDI dialog. If the item is in Accepted status, the cost change records are written to the cost change dialog. From there, when the cost change is approved, SCCEXT processes these cost changes and updates system costs

Cost change batch module descriptions

EDIUPCAT (Vendor item information upload) – This module uploads a flat file that originates as the output of a client’s EDI translation software application. The module then updates the EDI_NEW_ITEM and EDI_COST_LOC tables.

SCCEXT (Supplier cost change extract) – This module writes to the price history (PRICE_HIST) table and transaction-level stock ledger (TRAN_DATA) from the ITEM_LOC_SOH table. The costs on approved orders may also be updated if the recalculate order indicator is set to “Yes” for the item-supplier combination. The PREPOST batch module, with the sccext_post function, runs after SCCEXT to update the status of the cost change to “Extracted.”

CCPRG (Cost event purge) – This module runs after SCCEXT.PC to remove old cost changes from the system using the following criteria:

- the status of the cost change is “Delete,” “Canceled,” or “Extracted”
- the status of the price change is “Rejected,” and the effective date of the cost change has met the requirement for the number of days that rejected cost changes are held

Note: The number of days that rejected price changes are held is determined by a system option.

Summary of cost change and related batch modules

Module name	**Description**	**Dependencies on other modules** (run before or after)
SCCEXT	Selects supplier cost change records, which are set to go into effect the next business day, and updates the following RMS tables with the new cost: ITEM_SUPP_COUNTRY_BRACKET_COST (if the supplier is bracket costing) ITEM_SUP_COUNTRY_LOC (holds the current unit cost) ITEM_SUP ITEM_SUP_COUNTRY See Volume 4 of the RMS 10.0 Operations Guide for additional information about SCCEXT.	Run daily in Phase 3 of RMS' batch schedule. Run before RPLBLD.PC and VRPLLBD.PC
EDIUPCAT	Processes supplier cost change data from a flat file supplied by the client from its EDI translation software	Run daily in Phase 23 of RMS' batch schedule, or as needed
CCPRG	Purges old supplier cost changes	Run monthly, or as needed

Payment terms

Payment terms are supplier-related financial arrangement information that is published to the Retek Integration Bus (RIB), along with the supplier and the supplier address, from the financial system. Payment terms are the terms established for paying a supplier (for example, 2.5% for 30 days, 3.5% for 15 days, 1.5% monthly, and so on). RMS 10.1 subscribes to a payment terms message that is held on the RIB. After confirming the validity of the records enclosed within the message, RMS 10.1 updates its tables with the information.

This overview describes the following:

- A description of the specific data within the payment terms message.
- A summary of the steps that occur during the processing of the payment terms message.
- A summary of the payment terms message that shows its corresponding DTD and mapping document.
- The table in RMS 10.1 that is affected by the payment terms subscription message.

Payment terms message

RMS subscribes to a flat payment terms message that consists of a payment terms record. A payment terms record can only be created or updated. Previous payment terms records are neither deleted nor modified; they are rendered enabled/disabled through a flag associated with the active/inactive date.

RMS subscribes to the PayTermDesc payment terms message.

Data in the payment terms message that has primary significance to RMS includes:

- The number that uniquely identifies the payment terms.
- The alphanumeric representation of the payment terms name that acts as the payment terms code in the financial system.
- A description of the payment terms (for example, 2.5% 30 days).
- The number of days until payment is due.
- The date for assigning an active date to the payment terms.
- The date for assigning an inactive date to the payment terms.
- The number of days in which payment must be made in order to receive the discount.
- The percent of discount if payment is made within the specified time frame.

Payment terms message subscription

The payment terms message subscription process primarily consists of the two following PL/SQL packages:

RMSSUB_PAYTERMCRE and its CONSUME procedure serve as RMS' subscription API. A second package, RMSSUB_PTRM, holds functions that performs the following tasks on data contained in the message:

- Parse
- Process
- Insert
- Update
- Validate

Message subscription process

From a high-level perspective, the payment terms message subscription process primarily consists of the following steps:

1 The RMS external payment terms adapter (eWay) recognizes that a message with a payment terms name exists on the RIB. The RIB calls the first package, which serves as RMS' subscription API: RMSSUB_PAYTERMCRE.CONSUME. This package initially processes the message and the XML CLOB contained in the message.

2 Additional functions contained in RMSSUB_PTRM are called in order to parse out the payment terms to memory, process the data, validate the data, and insert the data into the TERMS table.

Payment terms message summary

The following table shows you the PayTermCre message, the document type definition (DTD) that describes the XML message, and the mapping document that describes the data contained in the message. The mapping document contains information that includes the source table, column, and data types. See the Retek 10.1 Integration Guide for more information regarding these documents.

Message Name	Type (DTD)	Mapping Document
PayTermCre	PayTermDesc.dtd	Map_PayTermDesc.xls

Payment terms table

The following description is for the primary table in RMS that holds payment terms data:

TERMS

This table contains one row for each set of supplier payment terms allowed within the company.

Security: location, product, price zone

RMS customers can take advantage of a security feature that defines which users can select or update location, product, and zone data by function. Because all users have full access as soon as they are entered into RMS, this security feature gives the RMS system administrator the ability to disallow access as necessary. This overview describes the three areas to which user access can be limited, the process by which this occurs, and the respective batch processes for each area.

Product, location, and zone security exists to supplement the database security that the database 'owner' grants to RMS users. All three areas have functional subsets to which access is controlled. The following table lists the functional subsets for each area.

Security Areas and Functions		
Location	**Product**	**Price Zone**
Stores, warehouses for… • promotion • stock orders (allocations and transfers) • allocations to and from • shipments • orders • stock counts • ticket requests • inventory adjustments • returns to vendors • store • Sales Audit store day • Sales Audit store ACH (automated clearinghouse)	Items at any level in the merchandise hierarchy, like department, class, subclass, and so on for… • pricing • costing • promotion • clearance • transfer • allocations • orders • stock counts	Price and cost zones for locations (stores and warehouses) for… • pricing • clearances

Security setup process

The process by which you apply security for any of these functional areas is identical:

- Add users to RMS
- Add a security group
- Associate users to a security group
- Associate location, product, or zone level security to a group

Each area has its own security matrix form. Use the form to:

- Select a group (to which you have already associated users)
- Select the functional area to which security will apply
- Apply the remaining selections required for the specific matrix form and save it

The security restrictions that you set up at the group level are applied to all users that are linked to the group. Any changes that you make to the security settings become effective after the respective batch program runs to rebuild the security records.

To learn more about setting up security, see the online help and user's guide. To learn more about the three batch programs associated with security, see the next section.

If security rules conflict

Occasionally security rules for a single user may overlap, thereby causing a conflict. RMS is set up to resolve this by saying that a rule that has a smaller scope overrules any that has a broader scope. Here is an example of how this would work.

Suppose that a certain user is assigned to two groups. One group has no update capability for a given region, but the other group allows updating for a specific store within that region. For this user there is a rule conflict. RMS resolves the conflict by granting the user update capability for the store. Thus, the rule that affects the lowest level in the organization hierarchy (update capability for the store) is given precedence over the rule that denies access to that same store's entire region. For all other stores in the region, that rule would continue to apply to the user.

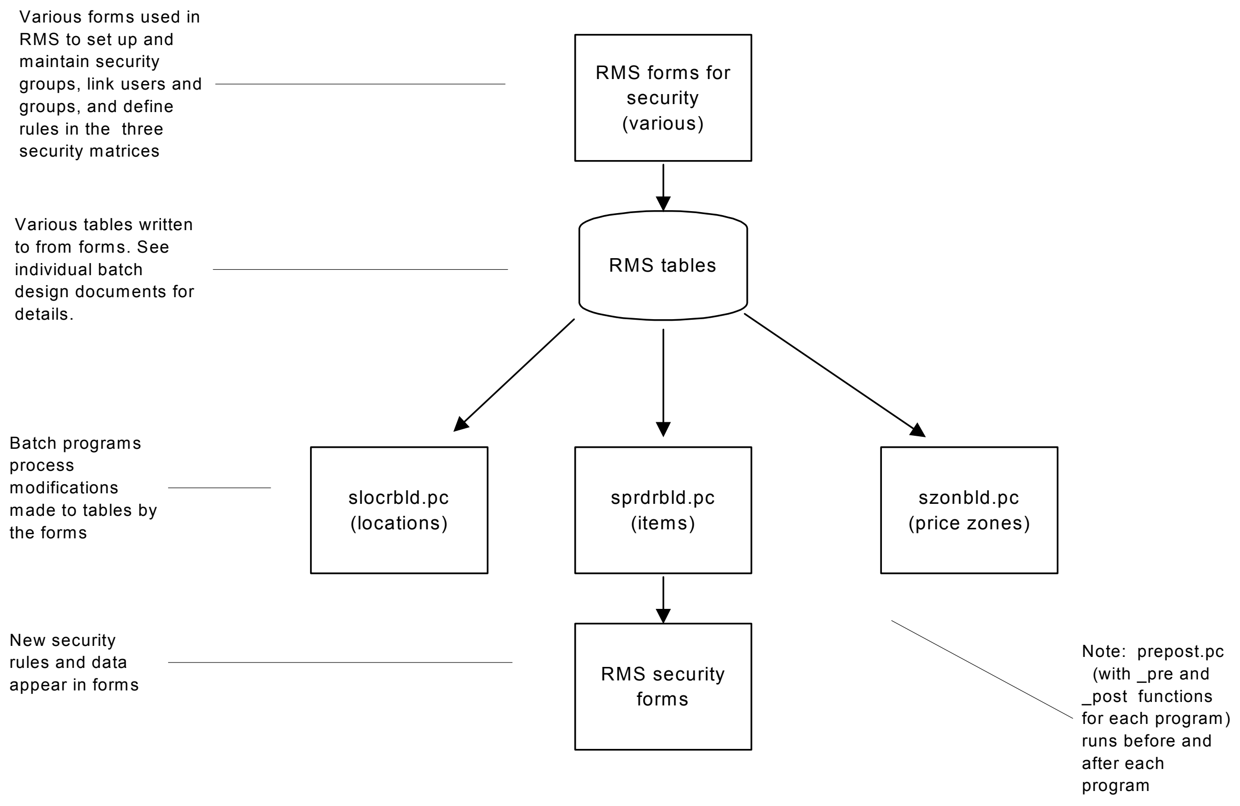
Security batch programs

Each of the three security areas has its own respective batch program that runs to update RMS tables whenever you add, modify, or delete a security group, user-group link, or security matrix form. Data does not become available for use until these programs have run within the daily batch-processing schedule. The three batch programs are:

- SLOCRBLD.PC
- SPRDRBLD.PC
- SZONRBLD.PC

The following diagram highlights the security process flow. Descriptions of each program follow.

Security Process Flow
(including batch programs)


Various forms used in RMS to set up and maintain security groups, link users and groups, and define rules in the three security matrices
RMS forms for security (various)
Various tables written to from forms. See individual batch design documents for details.
RMS tables
Batch programs process modifications made to tables by the forms
slocrbld.pc (locations)
sprdrbld.pc (items)
szonbld.pc (price zones)
New security rules and data appear in forms
RMS security forms
Note: prepost.pc (with _pre and _post functions for each program) runs before and after each program

SLOCRBLD.PC – location security rebuild

SLOCRBLD handles the maintenance for the location security data. Locations can have different update and select attributes for a given user that define if the user can select or update one or more locations as defined by the rules. The RMS table impacted by this program is SEC_USER_LOC_MATRIX.

SPRDRBLD.PC – product security rebuild

This program processes product security data. Items can have different update or select attributes for a given user for any of the item functional areas, such as pricing, orders, and allocations. The RMS table impacted by this program is SEC_USER_PROD_MATRIX.

SZONRBLD.PC – zone security rebuild

SZONRBLD.PC maintains zone security data. Users can have access to various price zones. The RMS table impacted by this program is SEC_USER_ZONE_MATRIX.

Prepost functions for security batch

In addition to the three batch programs that rebuild the security matrix tables, there is an RMS utility program called PREPOST.PC that runs before and after each of the security programs. PREPOST.PC is a program that performs a variety of functions, many of which simply truncate tables in preparation for table updates or switch an indicator on a table. Such is the case with all of the security programs. This section describes the pre- and post-functions performed by PREPOST.PC for the security programs.

'Pre-' functions of PREPOST.PC

Before SLOCRBLD.PC, SPRDRBLD.PC, or SZONRBLD.PC run, PREPOST.PC calls a 'pre' function named: truncate_user_sec_table()

This function truncates the user security matrix tables and prepares for the security rebuild that is specific to the security batch program that you plan to run. For example, if you plan to run the product security batch program SPRDRBLD.PC, the function truncates the table that is affected by SPRDRBLD.PC, thereby preparing the table for rebuild when SPRDRBLD.PC itself runs.

Were you planning to run SLOCRBLD.PC instead, then the function would truncate the table specific to the location security rebuild.

'Post-' functions of PREPOST.PC

Similar to the 'pre' function, PREPOST.PC also has a 'post' function named: update_security_indicator()

This function resets a security update indicator after you run any of the three security rebuild programs, preparing RMS tables for the next batch run updates by these programs.

A note about the product rebuild

Unlike the location and zone rebuild processes, the product rebuild uses an additional step after SPRDRBLD.PC runs and before PREPOST.PC's post function. Because of the large amount of data processed by SPRDRBLD.PC, the program outputs a flat file. The data in the file flat is then loaded into the security user product matrix by an Oracle SQL*Load process.

Security programs in the batch schedule

Whenever you run the batch schedule, you begin and end the run with two additional PREPOST.PC functions. At the beginning of the schedule, you run:

btchcycl pre

This function disables all security thus ensuring that the batch programs can access all required data. Disabling security also speeds performance of the schedule. At the end of the batch schedule, run this PREPOST.PC function to again enable security:

btchcycl post

Batch modules for location, product, and zone security

This table lists all batch modules that are involved in RMS security table updates.

Security Batch Programs			
Batch Module Program Name	**What It Does**	**When to Run It**	**Run Before/After Other Modules?**

slocrbld.pc	Updates security tables for stores and warehouses	Daily, Phase4	After prepost.pc with slocrbld_pre function:
			truncate_user_sec_table()
			Before prepost.pc with slocrbld_post function:
			update_security_indicator()

sprdrbld.pc	Updates security tables for products (items within the merchandise hierarchy)	Daily, Phase 4	After prepost.pc with sprdrbld_pre function: truncate_user_sec_table() Before a SQL*Load process runs to load the program's flat file data into the security user product matrix Before prepost.pc with sprdrbld_post function: update_security_indicator()

szonrbld.pc	Updates security tables for price and cost data for locations	Ad Hoc	After prepost.pc with szonrbld_pre function:
			truncate_user_sec_table()
			Before prepost.pc with szonrbld_post function:
			update_security_indicator()

Supplier

RMS 10.1 subscribes to supplier and supplier address data that is published from an external financial application and publishes supplier and address data to other external applications. This overview describes:

- The supplier subscription process
- The supplier publication process
- The nature of the subscribed and published messages
- The primary tables that serve as targets and sources of data contained in messages
- One supplier related batch (Pro*C) module–SUPMTH– that runs within RMS' batch processing schedule

Supplier message subscription

The supplier and supplier address message subscription process primarily consists of the two following PL/SQL packages:

RMSSUB_VENDORCRE and its CONSUME procedure serves as RMS' subscription API. A second package, RMSSUB_SUPPLIERS, holds functions that performs the following tasks on data contained in the message:

- parse
- process
- insert
- update
- validate

Message subscription process

1. The RMS supplier adapter (eWay) recognizes that a supplier or supplier address message exists on the RIB.
2. For a new supplier and address message, the adapter calls RMSSUB_VENDORCRE.CONSUME to initially process the message and the XML CLOB contained in the message.
3. Additional functions contained in RMSSUB_SUPPLIERS are called in order to parse out the supplier or address to memory, process the data, validate the data, and insert the data into the SUPS or ADDR tables, for supplier and address records respectively. Processing includes a check for the appropriate financial application in RMS on the SYSTEM_OPTIONS table's FINANCIAL_AP column.

Supplier publication

RMS 10.0 publishes supplier and supplier address data messages to subscribing applications so that those applications are able to keep their vendor tables current with RMS. This overview focuses on:

- RMS vendor event messages, from their source tables, through message creation, to final publication, to the Retek Integration Bus (RIB)
- One supplier related batch (Pro*C) program–SUPMTH.PC– that runs within RMS' batch processing schedule

Supplier and address tables, event triggers, and messages

The RMS supplier and supplier address tables hold data at the base level within RMS. One additional message family manager queue table serves as the staging table for both supplier and address that are produced for publication to the RIB. An event on a base table causes that data to be populated on the respective queue. The following are brief descriptions of all three tables:

SUPS – This table contains one row for each supplier.

ADDR – This table contains one row for each supplier or partner address. The SEQ_NO column is required because multiple addresses can exist for each address type. Only these valid address types from the table are published:

- Returns
- Order
- Invoice

SUPPLIER_MFQUEUE – This is the message queue that keeps track of all message events that occur on the supplier (SUPS) and addresses (ADDR) tables.

Detailed descriptions of these tables are in the RMS 10.0 Data Model document.

Event triggers

The SUPS and ADDR tables hold triggers for each row, or record, on the respective table. Any time that an event occurs on a table–that is, an insertion of a record, update to an existing record, or deletion of a record–the appropriate trigger 'fires' to begin the message creation process.

- The trigger for the SUPS table is **EC_TABLE_SUP_AIUDR**.
- The trigger for the ADDR table is **EC_TABLE_ADR_AIUDR**.

The next section describes each trigger.

Trigger descriptions

EC_TABLE_SUP_AIUDR –

1 This trigger captures inserts, updates, and deletes to the SUPS table.

2 The trigger writes data into the SUPPLIER_MFQUEUE message queue.

3 The trigger calls SUPPLIER_XML.BUILD_SUPPLIER to create the XML message.

4 The trigger calls RMSMFM_SUPPLIER.ADDTOQ to insert the message into the message queue.

EC_TABLE_ADR_AIUDR –

1 This trigger captures inserts, updates, and deletes to the ADDR table.

2 The trigger writes data into the supplier_mfqueue message queue.

3 The trigger calls SUPPLIER _XML.BUILD_SUPPLIER to create the XML message.

4 The trigger calls RMSMFM_SUPPLIER.ADDTOQ to insert the message into the message queue.

Supplier messages

There are six messages that pertain to the supplier message family, three for the supplier and three for addresses. Here are the supplier message short names:

- VendorCre
- VendorHdrMod
- VendorDel
- VendorAddrCre
- VendorAddrMod
- VendorAddrDel

Message family manager and queue

This section describes the message family manager (MFM) for suppliers.

RMSMFM_SUPPLIER – This MFM inserts and retrieves messages from the message queue. It contains the public procedures ADDTOQ, which inserts a message into the message queue, and GETNXT, which retrieves the next message on the message queue.

Message summary

The following table lists each supplier message by its message type that appears on the queue table, the document type definition (DTD) that describes the XML message, and the mapping document that describes the data contained in the message. Consult the Retek 10 Integration Guide to view these documents.

Message Type	Type (DTD)	Mapping Document
VendorCre	VendorDesc.dtd	Map_VendorDesc.xls
VendorHdrMod	VendorHdrDesc.dtd	Map_VendorHdrDesc.xls
VendorDel	VendorRef.dtd	Map_VendorRef.xls
VendorAddrCre	VendorAddrDesc.dtd	Map_VendorAddrDesc.xls
VendorAddrMod	VendorAddrDesc.dtd	Map_VendorAddrDesc.xls
VendorAddrDel	VendorAddrRef.dtd	Map_VendorAddrRef.xls

Message creation and publishing process

The message family manager inserts messages on the queue and marks each one with a sequence number. The goal is continue inserting new messages and replacing lower sequenced number messages of the same type until certain parameters are met. The private procedure CAN_CREATE determines if a complete hierarchical supplier message can be created based upon the existence of correct address types and additional flags that must be set.

Batch module SUPMTH

The Supplier Data Amount Repository (SUPMTH) module is executed based on multiple transaction types for each department-supplier combination in the system. Its primary function is to convert daily transaction data to monthly data. After all data are converted, the daily information is deleted to reset the system for the next period by the batch module PREPOST and its supmth_post function.

The Supplier Data Amount Repository (supmth) module should be run during Phase 3 of the RMS batch schedule, on a monthly basis.

Value added tax maintenance

Value-added tax (VAT) functionality is optional in RMS. In several countries, value added taxes (VAT) must be considered when determining the monetary value of items. VAT amounts appear in several modules of the system, such as purchase orders, pricing, contracts, stock ledger, and invoice matching. This overview describes the RMS system settings that impact VAT, along with the batch module VATDLXPL that associates items with a given VAT region and VAT code.

Value added tax rates are identified by VAT code. When VAT codes are associated with a VAT region, they are assigned a VAT type. The VAT type indicates that the tax rate is used in one of the following types of calculations:

- Cost: The tax rate is applied to purchase transactions.
- Retail: The tax rate is applied to sales transactions.
- Both: The tax rate is applied to purchase and sales transactions.

Value added taxes are reflected in the stock ledger when 1) the retail method of accounting is used and 2) the system is set up to include VAT in retail calculations.

A number of the system settings in RMS, which are described beginning in the next section, indicate how you wish to implement VAT.

System level VAT

The VAT_IND column on the SYSTEM_OPTIONS table is the primary means to initiate VAT in RMS. By entering “Y” in this column, you are telling RMS that you want to include VAT in the system.

System class level VAT

The CLASS_LEVEL_VAT_IND column on the SYSTEM_OPTIONS table allows you to include or exclude VAT at the class level of the merchandise hierarchy. Enter “Y” in this column to manage VAT inclusion or exclusion from retail at the class level. Enter “N” in this column if you do not want to manage VAT at the class level. Entering “N” will mean that VAT is included in the retail price in RMS and in the point-of-sale (POS) download for all classes. The POS upload process is controlled by the store VAT indicator, which is described later in this overview.

Department VAT

The department table (DEPS) holds the DEPT_VAT_INCL_IND column that is used to enable or disable VAT in retail prices for all classes in the department. This indicator is used only to default to the class level indicator when classes are initially set up for the department and is only available when the system level class VAT option is on. When VAT is turned on in the system and not defined at the class level, this field defaults to “Y”. When VAT is turned off in the system, this field defaults to “N”.

Class VAT

The CLASS_VAT_IND column on the CLASS table determines if retail is displayed and held with or without VAT. The default setting is inherited from the class's department (see the preceding section). You can edit the value in this column only when VAT is turned on in the system and defined at the class level.

By entering "Y" in this column, you are saying that you want VAT included in the retail price for all items in that class. Both point-of-sale (POS) download (POSDNLD) and POS upload (POSUPLD) will include VAT in the retail price.

By entering "N" in this column, you are saying that you want to exclude VAT from point-of-sale (POS) download (POSDNLD) and POS upload (POSUPLD) of retail prices for the entire class.

Instructions that are sent to allow the POS to add VAT are contained in these columns on the POS_MODS table:

- Vat_code – code for the VAT rate
- Vat_rate – the actual rate referenced by the VAT code
- Class_vat_ind

Store VAT indicator

If you select "N" in the CLASS_LEVEL_VAT_IND column on the SYSTEM_OPTIONS table, you can still choose VAT settings for a store. The VAT_INCLUDE_IND column on the STORE table allows you to include or exclude VAT at the store for POS upload only.

Enter "Y" in this column to always include VAT in the retail price in the POS upload process. Enter "N" to exclude VAT from POS uploaded prices.

Send VAT rate to POS

VAT rates are sent through the POS to the store and are contained in these columns on the POS_MODS table:

- Vat_code – code for the VAT rate
- Vat_rate – the actual rate referenced by the VAT code
- Class_vat_ind

Special note: retail method stock ledger and VAT

If the stock ledger for a department is set to use the retail method of accounting, an additional setting is required to ensure that VAT is, or is not, included in retail values. The STKLDGR_VAT_INCL_RETL_IND column (SYSTEM_OPTIONS table) value of "Y" results in all retail values in the stock ledger (sales retail, purchase retail, gross margin, and so on) being VAT inclusive. "N" indicates that VAT is excluded from retail values.

Batch module – VATDLXPL

The value-added tax rate maintenance module updates VAT information for each item associated with a given VAT region and VAT code. Run the module as needed; however, it must be run in Phase 1 of the batch schedule, before any pricing modules are executed.

Replenishment

Replenishment batch module components are designed to manage stock levels, by using stock order allocations. For RMS 10.1, only replenishment and Retek Allocation can create stock order allocations. This overview describes batch functionality for replenishment, including investment buy, along with descriptions of the major tables involved in the replenishment process.

Replenishment process

Replenishment operates in this sequence:

1 Build the purchase order

2 Scale the order

3 Split the order among trucks

4 Compare approved replenishment orders against applicable vendor minimums and reset back to 'W'orksheet status those orders that do not meet minimum quantities

Summary of replenishment batch modules

Replenishment batch module name	**Description**	**Dependencies on other modules** (run before or after)
SOUPLD	Processes store order data from an external system flat file that are used later in the replenishment process to generate recommended order quantities. Accepts an input file that contains: • item to be ordered • store requesting the item • needed quantity in eaches, cases, or pallets (later converted to standard unit of measurements) • need date Module validates that item and store are on replenishment with a replenishment method of "Store Orders" (REPL_ITEM_LOC.REPL_METHOD).	Run daily in Phase 2 of RMS' batch schedule Run before all replenishment and investment buy batch modules.

Replenishment batch module name	**Description**	**Dependencies on other modules** (run before or after)
RPLATUPD	Maintains replenishment attributes for an item list by calling the package REPL_ATTRIBUTE_MAINTENANCE_SQL (rplattrb/s.pls) to write changes to the tables REPL_ATTR_UPDATE_ITEM and REPL_ATTR_UPDATE_LOC that are initiated by the replenishment attribute form.	Run daily in Phase 3 of RMS' batch schedule. Run after the batch module PREPOST with the RPLATUPD_PRE argument. Run before the replenishment batch programs, RPLADJ, RPLEXT, and REQEXT. Run before the batch module PREPOST with the argument RPLATUPD_POST.
RILMAINT	Processes replenishment attributes from the REPL_ITEM_LOC_UPDATES table to the REPL_ITEM_LOC table.	Run in Phase 3 of RMS' batch schedule. Run after RMS batch modules STOREADD and RPLATUPD. Run before the batch module PREPOST using the argument RILMAINT_POST Run before the batch module RPLADJ.
RPLADJ	Recalculates the maximum stock levels for all item-location combinations with a replenishment method of 'F' (floating point) and populates the table REPL_ITEM_LOC. The floating model stock method will dynamically calculate an order-up-to-level. The maximum model stock is calculated using the sales history of various periods of time in order to accommodate seasonality as well as trend. The sales history is obtained from the item_loc_hist table	Run after the batch module RPLATUPD. Run before the RMS batch modules RPLEXT and REQEXT.

Replenishment batch module name	Description	Dependencies on other modules (run before or after)
REQEXT	Cycles through every item-location combination that is set to be reviewed on the current day and calculates the quantity of the item that needs to be transferred to the location. Transfers are created and records are written to the Replenishment Results (REPL_RESULTS) table depending on how the order control parameter is set at the item-location level.	Run in Phase 3 of RMS' batch schedule. Run after the batch modules, RPLATUPD, RPLADJ (that update replenishment calculation attributes). Run before PREPOST with the REQEXT_POST function. Run before RPLEXT.
RPLEXT	Calculates item quantities to be ordered for a location. Writes temporary orders to the tables ORD_TEMP, when automatic order creation is enabled (semi-automatic and automatic order control), and REPL_RESULTS. ORD_TEMP is later reviewed by the module CNTRPRSS in its evaluation of orders against contract types A, C, and D.	Run daily in Phase 3 of RMS' batch schedule. Run after PREPOST with the RPL_PRE function. Run before the batch module CNTRPRSS.
CNTRPRSS	Evaluates contract and supplier information of A, C, and D type contracts against recommended order quantities created by the RPLEXT module on the ORD_TEMP table. Suggests the best available contract for each item. Updates the REPL_RESULTS and ORD_TEMP tables to hold information about the quantity of the item that is satisfied by the contract.	Run daily in Phase 3 of RMS' batch schedule. Run after RPLEXT. Run before RPLBLD.

Replenishment batch module name	**Description**	**Dependencies on other modules** (run before or after)
IBEXPL	Determines inventory buy eligibility that is set at one of these levels: • Supplier-department-location • Supplier-location (warehouse locations only) • Supplier-department • Supplier Applies investment buy values that are defined on the SUP_INV_MGMT or WH_DEPT tables as applicable. If no values exist on the tables, this module accepts the default values held on the SYSTEM_OPTIONS table. (See section "Investment buy system options" later in this overview.)	Run daily in Phase 3 of the RMS batch schedule. Run after RMS batch modules RPLEXT. Run before the module IBCALC.
IBCALC	Calculates investment buy opportunities and writes the resulting recommended order quantities (ROQ) to the IB_RESULTS table.	Run before the module RPLBLD.
RPLBLD	Builds purchase orders from Recommended Order Quantities (ROQ) located on the ORD_TEMP table (populated by the RPLEXT module), and on the IB_RESULTS table (populated by the IBCALC module). Calls the order library (ORDLIB.h) to apply order creation logic.	Run daily in Phase 3 of RMS' batch schedule. Run after RMS batch modules RPLEXT and CNTPRSS (if contracts are used).
SUPCNSTR	Scales eligible orders during the nightly replenishment run.	Run daily in Phase 3 of RMS' batch schedule. Run after RMS batch modules RPLBLD. Run before RMS batch module RPLPRG.

Replenishment batch module name	Description	Dependencies on other modules (run before or after)
RPLSPLIT	Calls the order library (ORDLIB.h) to provide truck-splitting processing, and creates new orders.	Run daily in Phase 3 of RMS' batch schedule. Run after RMS batch module SUPCNSTR. Run before the RPLAPPRV module.
RPLAPPRV	Compares all approved replenishment orders created during the nightly batch run with any vendor minimums that may exist. Orders that do not meet the vendor minimums are either deleted or placed in Worksheet status.	Run in Phase 3 of RMS' batch schedule. Run after the batch module RPLSPLIT.
RPLPRG	Purges the following tables of dated rows. Values are held for each table in the SYSTEM_OPTIONS table. The SYSTEM_OPTIONS column that holds the number of days value is listed in parentheses: REPL_RESULTS (REPL_RESULTS_PURGE_DAYS) STORE_ORDERS (STORE_ORDERS_PURGE_DAYS) IB_RESULTS (IB_RESULTS_PURGE_DAYS)	Run as needed.

Primary replenishment tables

The following descriptions are for the primary replenishment and investment buy tables in RMS. It is not a complete list of all tables that are involved in the replenishment process:

REPL_ITEM_LOC – This table holds item-location replenishment attributes such as review cycle and activation dates. Note in particular the column REPL_METHOD that contains the code value that the modules REQEXT and RPLEXT use to calculate the recommended order quantities for the item-location. Replenishment method values include the following:

- C – Constant
- M – Minimum-Maximum
- F – Floating Point
- T – Time Supply (used with forecasting)
- Time Supply Seasonal (used with forecasting)
- Time Supply Issues (used with forecasting)
- D – Dynamic (used with forecasting)
- Dynamic Seasonal (used with forecasting)
- Dynamic Issues (used with forecasting)
- SO – Store Orders.

REPL_ITEM_LOC.LAST_ROQ – This column on the REPL_ITEM_LOC table contains the last recommended order quantity created by the vendor replenishment extraction module RPLEXT. The ROQ value is used by the investment buy opportunity calculation module (IBCALC) to calculate future available quantity for the item-location combination.

See Also: The RMS 10.0 Data Model for a complete description of the REPL_ITEM_LOC table.

REPL_ITEM_LOC.REPL_ORDER_CTRL – Determines if the replenishment process creates an actual order or transfer line item for the item-location if there is a need for the item-location or if only a record is written to the replenishment results table. Valid values are:

- ‘M’anual (a record is written to the Replenishment Results table – no order/transfer line item is created)
- ‘S’emi-Automatic (an order/transfer line item is created - the order line item will be added to an order in Worksheet status, the transfer line item will be added to a transfer in ‘S’ubmitted status with a freight type of Normal)
- ‘A’utomatic (an order/transfer line item is created - the order line item will be added to an order in Approved status, the transfer line item will be added to a transfer in Approved status with a freight type of Normal)

- ‘B’uyer Worksheet (a record is written to the Replenishment Results table and can be added to a purchase order on the Buyer Worksheet. A transfer line item is added to a transfer in ‘S’ubmitted status with a freight type of Normal.)

REPL_RESULTS – This table is used to store item location level replenishment results information and the replenishment attributes used to drive the order quantities for the item location.

ORD_TEMP – This table is used during the automatic replenishment cycle to temporarily store order line items generated during batch RPLXT. The actual orders are then created later in the batch run by consolidating these line items by department/supplier/delivery location (store/warehouse).

IB_RESULTS – This table contains investment buy recommended order quantities (ROQ) for an item-supplier-country-location (warehouse) along with the specific factors that lead to the ROQ. It contains the actual order quantity (AOQ), which may have been modified by the user. If the investment buy quantity is placed on the purchase order, the order number appears on the table.

ORD_INV_MGMT – Determines whether the stock out comparisons for ‘Due’ order determination should be performed in units (standard unit of measure), cost, or profit (that is, retail - cost) in the order’s currency. It is only used for replenishment orders when the Due Order Indicator is set to Yes. Valid values include:

- U – Unit service basis. Stock out amounts calculated in units (standard unit of measures).
- C – Cost service basis. Stock out amounts calculated as the stock out in units multiplied by the item's cost.
- P – Profit service basis. Stock out amounts calculated as the stock out in units multiplied by the item's margin (that is, retail - cost).

This table also holds a number of scaling and truck splitting parameters.

Investment buy

Investment buy facilitates the process of purchasing inventory in excess of the replenishment recommendation in order to take advantage of a supplier deal or to leverage inventory against a cost increase. The inventory is stored at the warehouse or in outside storage to be used for future issues to the stores. The recommended quantity to ‘investment buy’ (that is, to order), is calculated based on the following:

- Amount of the deal or cost increase
- Upcoming deals for the product
- Cost of money
- Cost of storage
- Forecasted demand for the product, using warehouse issue values calculated by Retek Demand Forecasting
- Target return on investment (ROI)

The rationale is to purchase as much product as profitable at the lower cost and to retain this profit rather than passing the discount on to customers and stores. The determination of how much product is profitable to purchase is based on the cost savings of the product versus the costs to purchase, store and handle the additional inventory.

Investment buy eligibility and order control are set at one of these four levels:

- Supplier
- Supplier-department
- Supplier-location (warehouse locations only)
- Supplier-department-location

Warehouses must be enabled for both replenishment and investment buy on RMS' WH (warehouse) table. In a multi-channel environment, virtual warehouses are linked to the physical warehouse.

The investment buy opportunity calculation takes place nightly during the batch run, after the replenishment need determination, but before the replenishment order build. The investment buy module IBCALC attempts to purchase additional inventory beyond the replenishment recommendation in order to achieve future cost savings. Two distinct events provide the incentive to purchase investment buy quantities:

- A current supplier deal ends within the look-ahead period.
- A future cost increase becomes active within the look-ahead period.

The calculation determines the future cost for a given item-supplier-country-location for physical warehouse locations only.

If the order control for a particular line item is 'buyer worksheet', it may be modified in the buyer worksheet dialog, and can be added to either new or existing purchase orders.

Investment buy system options

The following columns are held on the SYSTEM_OPTIONS table for investment buy:

- LOOK_AHEAD_DAYS – The number of days before a cost event (end of a deal, or a cost increase) that the investment buy opportunity begins to calculate an event
- COST_WH_STORAGE – Contains the default cost of warehouse storage, expressed as the weekly cost based on the unit of measure specified in this table's COST_WH_STORAGE_UOM column. This value is held in the primary system currency. You can change this value at the warehouse or warehouse-department level.
- COST_OUT_STORAGE – Contains the default cost of outside storage, expressed as the weekly cost base on the unit of measure specified in COST_OUT_STORAGE_UOM. This value is held in the primary system currency. You can change this value at the warehouse or warehouse-department level.

- COST_LEVEL – Indicates which cost bucket is used when calculating the return on investment for investment buy opportunities. Valid values are 'N' for net cost, 'NN' for net net cost and 'DNN' for dead net net cost.
- STORAGE_TYPE – Indicates which type of storage cost should be used as the default storage cost when calculating investment buy opportunities. Valid values are 'W'arehouse and 'O'utside. You can change this value at the warehouse or warehouse-department level.
- MAX_WEEKS_SUPPLY – Contains the default maximum weeks of supply to use in the investment buy opportunity calculation. The calculation does not recommend an order quantity that would stock the associated location (currently warehouses only) for a period beyond this number of weeks. You can change this value at the warehouse or warehouse-department level.
- TARGET_ROI – Contains the default return on investment that must be met or exceeded for the investment buy opportunity to recommend an order quantity. You can change this value at the warehouse or warehouse-department level.
- IB_RESULTS_PURGE_DAYS – Contains the number of days that records on the investment buy results table (IB_RESULTS) should be kept before being purged. If an investment buy result record's create_date plus this value is equal to or beyond the current system date, the record is deleted by the PREPOST batch module prior to the investment buy opportunity calculation.

See Also: The RMS 10.0 Data Model for a complete description of the SYSTEM_OPTIONS table and the investment buy columns.

Volume 2 – Message publication and subscription designs

Item Message Family Manager Publishing Design

Functional Area

Items

Design Overview

The item message family manager is a package of procedures that adds item family messages to the item queue and publishes these messages for the integration bus to route. Triggers on all the item family tables call a procedure from this package to add a “create”, “modify” or “delete” message to the queue. The integration bus calls a procedure in this package to retrieve the next publishable item message from the queue.

All the components that comprise the creation of an item, the item/supplier for example, remain in the queue until the item approval modification message has been published. Any modifications or deletions that occur between item creation in “W”(worksheet) status and “A”(Approved) status are applied to the “create” messages or deleted from the queue as required. For example, if an item UDA is added before item approval and then later deleted before item approval, the item UDA “create” message would be deleted from the queue before publishing the item. If an item/supplier record is updated for a new item before the item is approved, the “create” message for that item/supplier is updated with the new data before the item is published. When the “modify” message that contains the “A”(Approved) status is the next record on the queue, the procedure formats a hierarchical message that contains the item header information and all the child detail records to pass to the integration bus.

Additions, modifications and deletions to item family records for existing approved items are published in the order that they are placed on the queue.

Unless otherwise noted, item publishing includes most of the columns from the item_master table and all of the item family child tables included in the publishing message. Sometimes only certain columns are published, and sometimes additional data is published with the column data from the table row. The item publishing message is built from the following tables:

Family Header

item_master - transaction level items only

descriptions for the code values
names for department, class and subclass
diff types
base retail price

Item Family Child Tables

item_supplier

item_supp_country

item_supp_country_dim

descriptions for the code values

item_master - reference items

item, item_number_type, item_parent, primary_ref_ind, format_id, prefix

packitem_breakout

pack_no, item, packitem_qty

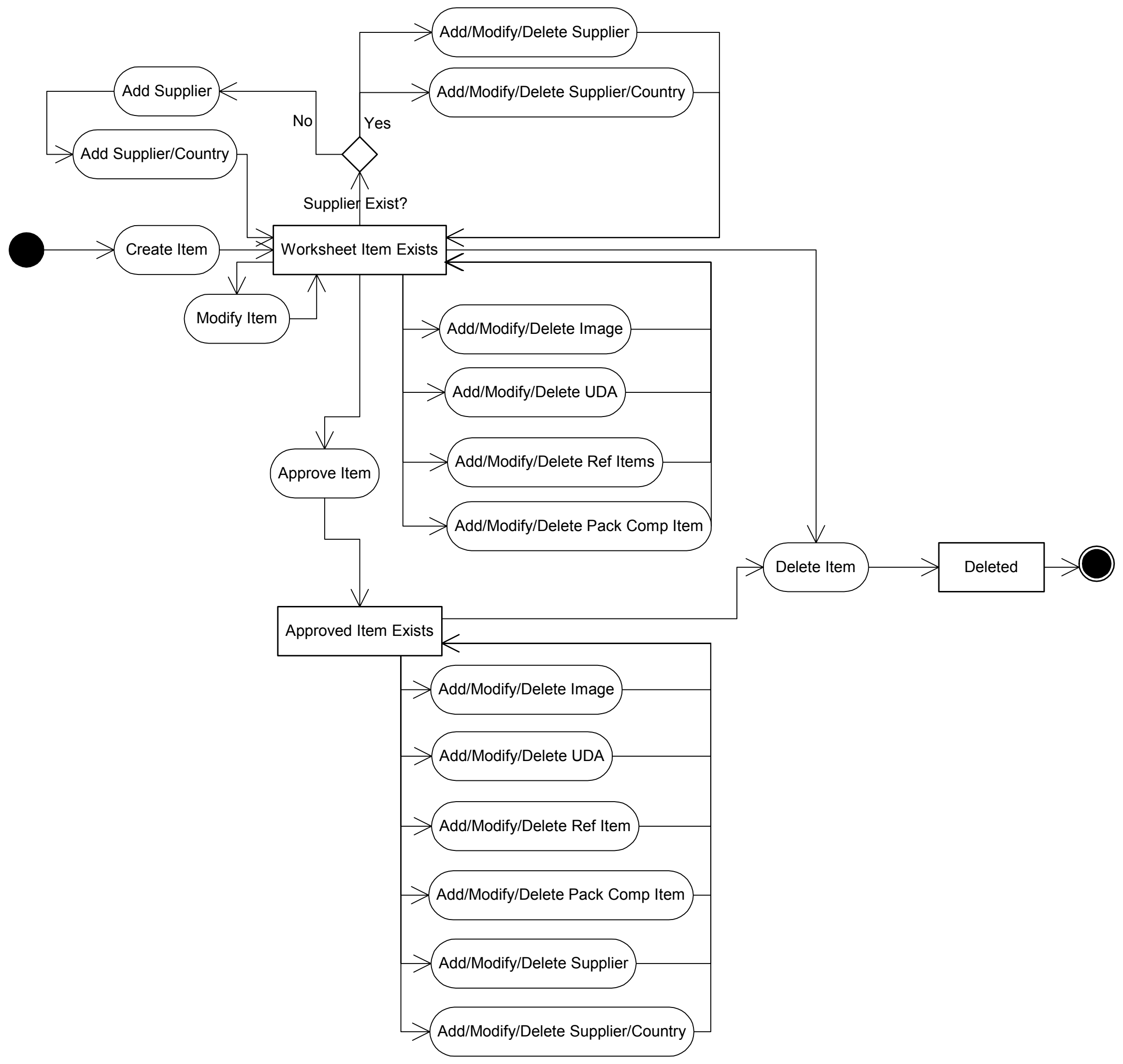
item_image

uda_item_ff

uda_item_lov

uda_item_date

State Diagram


Add/Modify/Delete Supplier
Add/Modify/Delete Supplier/Country
Add Supplier
Add Supplier/Country
No
Yes
Supplier Exist?
Create Item
Worksheet Item Exists
Modify Item
Add/Modify/Delete Image
Add/Modify/Delete UDA
Add/Modify/Delete Ref Items
Add/Modify/Delete Pack Comp Item
Approve Item
Delete Item
Deleted
Approved Item Exists
Add/Modify/Delete Image
Add/Modify/Delete UDA
Add/Modify/Delete Ref Item
Add/Modify/Delete Pack Comp Item
Add/Modify/Delete Supplier
Add/Modify/Delete Supplier/Country

Description of Activities

Create a Worksheet Item

1. **Prerequisites**: No prerequisites exist for creating an item except that RMS foundation data such as departments and suppliers exist first. Items are created using the RMS online item dialogue.
2. **Activity Detail**: The creation of the item is the first step of gathering all the hierarchical information needed for publishing the item.
3. **Messages**: A message for the item creation is placed on the queue for future publishing. This is a flat message that will be collected with the item detail messages to comprise the final hierarchical message. It will not be published until the item is approved. The presence of this message on the queue signals the publishing process that more detail information for the item is forthcoming.

Approve an Item

1. **Prerequisites**: An item must exist and be submitted for approval.
2. **Activity Detail**: The item record is updated in the item_master table. An ItemHdrMod message type record is inserted on the item_mfqueue table. Once the item is approved, it is of interest to other software systems. It can be included in orders, transfers, shipments, etc.
3. **Messages**: ItemHdrDesc message type is created. It is a flat, synchronous message containing this item record. The approved item create message is a hierarchical message containing the item and all the item family detail records. First an ItemDesc node is created and the ItemHdrDesc message is added to this message. Next, all the child messages are appended to the message until there are no more records in the item_mfqueue table for this item. Then the final item message is formatted.

Modify an Item

1. **Prerequisites**: An item can have any status to be modified. Once the item is approved, there are only a few fields that can be modified.
2. **Activity Detail**: The item record is updated in the item_master table. An ItemHdrMod message type record is inserted on the item_mfqueue table.
3. **Messages**: ItemHdrDesc message type is created. It is a flat, synchronous message containing this item record. If a record that has an ItemCre message type exists on the item_mfqueue table for this item, this “modify” message is never used in publishing. Only the final “modify” item record message with an ‘A’(Approved) status is published. If no ItemCre record exists on the item_mfqueue table for this item, it is published as a flat message.

Create Item/Supplier

1 **Prerequisites**: The supplier and the item already exist.

2 **Activity Detail**: The item/supplier combination is inserted into the item_supplier table. An ItemSupCre message type record is also inserted on the item_mfqueue table.

3 **Messages**: ItemSupDesc message type is created. It is a flat, synchronous message containing this item/supplier record. It is published as part of the item creation message when the item is approved, or flat after the original publication of the item creation.

Modify Item/Supplier

1 **Prerequisites**: The item/supplier combination already exists.

2 **Activity Detail**: The item/supplier record is updated in the item_supplier table. An ItemSupMod message type record is inserted on the item_mfqueue table.

3 **Messages**: ItemSupDesc message type is created. It is a flat, synchronous message containing this item/supplier record. If records that have an ItemCre and an ItemSupCre message type exist on the item_mfqueue table for this item/supplier, the message is updated with the "modify" message and published as part of the item creation message when the item is approved. Otherwise, it is published as a flat message.

Delete Item/Supplier

1 **Prerequisites**: The item/supplier combination already exists and is not being used somewhere in the system.

2 **Activity Detail**: The item/supplier record is deleted from the item_supplier table and all child records from the item_supp_country and item_supp_country_dim tables. An ItemSupDel message type record is inserted on the item_mfqueue table.

3 **Messages**: ItemSupRef message type is created. It is a flat, synchronous message containing the keys for this item/supplier record. If records that have an ItemCre and an ItemSupCre message type exist on the item_mfqueue table for this item/supplier, the ItemSupCre and any ItemSupMod records are deleted from the item_mfqueue table. Otherwise, it is published as a flat message.

Create Item/Supplier/Country

1 **Prerequisites**: The supplier, country and the item already exist.

2 **Activity Detail**: The item/supplier/country combination is inserted into the item_supp_country table. An ItemSupCtyCre message type record is also inserted on the item_mfqueue table.

3 **Messages**: ItemSupCtyDesc message type is created. It is a flat, synchronous message containing this item/supplier/country record. It is published as part of the item creation message when the item is approved, or flat after the original publication of the item creation.

Modify Item/Supplier/Country

1 **Prerequisites**: The item/supplier/country combination already exists.

2 **Activity Detail**: The item/supplier/country record is updated in the item_supp_country table. An ItemSupCtyMod message type record is inserted on the item_mfqueue table.

3 **Messages**: ItemSupCtyDesc message type is created. It is a flat, synchronous message containing this item/supplier/country record. If records that have an ItemCre and an ItemSupCtyCre message type exist on the item_mfqueue table for this item/supplier/country, the message is updated with the “modify” message and published as part of the item creation message when the item is approved. Otherwise, it is published as a flat message.

Delete Item/Supplier/Country

1 **Prerequisites**: The item/supplier/country combination already exists.

2 **Activity Detail**: The item/supplier/country record is deleted from the item_supp_country table and all child records from the item_supp_country_dim table. An ItemSupCtyDel message type record is inserted on the item_mfqueue table.

3 **Messages**: ItemSupCtyRef message type is created. It is a flat, synchronous message containing the keys for this item/supplier/country record. If records that have an ItemCre and an ItemSupCtyCre message type exist on the item_mfqueue table for this item/supplier/country, the ItemSupCtyCre and any ItemSupCtyMod records are deleted from the item_mfqueue table. Otherwise, it is published as a flat message.

Create Item/Supplier/Country/Dimension

1 **Prerequisites**: The item/supplier/country already exists.

2 **Activity Detail**: The item/supplier/country/dimension combination is inserted into the item_supp_country_dim table. An ISCDimCre message type record is also inserted on the item_mfqueue table.

3 **Messages**: ISCDimDesc message type is created. It is a flat, synchronous message containing this item/supplier/country/dimension record. It is published as part of the item creation message when the item is approved, or flat after the original publication of the item creation.

Modify Item/Supplier/Country/Dimension

1 **Prerequisites**: The item/supplier/country/dimension combination already exists.

2 **Activity Detail**: The item/supplier/country/dimension record is updated in the item_supp_country_dim table. An ISCDimMod message type record is inserted on the item_mfqueue table.

3 **Messages**: ISCDimDesc message type is created. It is a flat, synchronous message containing this item/supplier record. If records that have an ItemCre and an ISCDimCre message type exist on the item_mfqueue table for this item/supplier/country/dimension, the message is updated with the "modify" message and published as part of the item creation message when the item is approved. Otherwise, it is published as a flat message.

Delete Item/Supplier/Country/Dimension

1 **Prerequisites**: The item/supplier/country/dimension combination already exists.

2 **Activity Detail**: The item/supplier/country/dimension record is deleted from the item_supp_country_dim table and all child records from the item_supp_country and item_supp_country_dim tables. An ISCDimDel message type record is inserted on the item_mfqueue table.

3 **Messages**: ISCDimRef message type is created. It is a flat, synchronous message containing the keys for this item/supplier/country/dimension record. If records that have an ItemCre and an ISCDimCre message type exist on the item_mfqueue table for this item/supplier/country/dimension, the ISCDimCre and any ISCDimMod records are deleted from the item_mfqueue table. Otherwise, it is published as a flat message.

CreateRef_Item

1 **Prerequisites**: The parent item exists.

2 **Activity Detail**: The item is inserted into the item_master table. An ItemUPCCre message type record is also inserted on the item_mfqueue table.

3 **Messages**: ItemUPCDesc message type is created. It is a flat, synchronous message containing this reference item record. It is published as part of the item creation message when the item is approved, or flat after the original publication of the item creation.

Modify Ref_Item

1 **Prerequisites**: The reference item exists as a child item.

2 **Activity Detail**: The item record is updated in the item_master table. An ItemUPCMod message type record is inserted on the item_mfqueue table.

3 **Messages**: ItemUPCDesc message type is created. It is a flat, synchronous message containing this reference item record. If records that have an ItemCre and an ItemUPCCre message type exist on the item_mfqueue table for this reference item, the message is updated with the "modify" message and published as part of the item creation message when the item is approved. Otherwise, it is published as a flat message.

Delete Ref_Item

1 **Prerequisites**: The reference item already exists as a child item.

2 **Activity Detail**: The reference item record is deleted from the item_master table. An ItemUPCDel message type record is inserted on the item_mfqueue table.

3 **Messages**: ItemUPCRef message type is created. It is a flat, synchronous message containing the keys for this reference item record. If records that have an ItemCre and an ItemUPCCre message type exist on the item_mfqueue table for this reference item, the ItemUPCCre and any ItemUPCMod records are deleted from the item_mfqueue table. Otherwise, it is published as a flat message.

Create Pack Comp

1 **Prerequisites**: The pack item exists.

2 **Activity Detail**: The pack comp is inserted into the packitem_breakout table. An ItemBOMCre message type record is also inserted on the item_mfqueue table.

3 **Messages**: ItemBOMDesc message type is created. It is a flat, synchronous message containing this pack comp record. It is published as part of the item creation message when the item is approved, or flat after the original publication of the item creation.

Modify Pack Comp

1 **Prerequisites**: The pack comp exists for the pack.

2 **Activity Detail**: The pack comp record is updated in the packitem_breakout table. An ItemBOMMod message type record is inserted on the item_mfqueue table.

3 **Messages**: ItemBOMDesc message type is created. It is a flat, synchronous message containing this pack comp record. If records that have an ItemCre and an ItemBOMCre message type exist on the item_mfqueue table for this pack comp, the message is updated with the “modify” message and published as part of the item creation message when the item is approved. Otherwise, it is published as a flat message.

Delete Pack Comp

1 **Prerequisites**: The pack comp already exists for the pack.

2 **Activity Detail**: The pack comp record is deleted from the packitem_breakout table. The packitem_qty is retrieved from the v_packitem_qty view. If the quantity for the pack comp is 0, an ItemBOMDel message type record is inserted on the item_mfqueue table. If the quantity for the pack comp greater than 0, an ItemBOMMod message type record is inserted on the item_mfqueue table.

3 **Messages**: If the message type is ItemBOMDel, a ItemBOMRef message type is created. It is a flat, synchronous message containing the keys for this pack comp record. If records that have an ItemCre and an ItemBOMCre message type exist on the item_mfqueue table for this pack comp, the ItemBOMCre and any ItemBOMMod records are deleted from the item_mfqueue table. Otherwise, it is published as a flat message. If the message type is ItemBOMMod, a message is create and processed as described in the Modify Pack Comp Messages section.

Create Item/Image

1 **Prerequisites**: The item already exists.

2 **Activity Detail**: The item/image combination is inserted into the item_image table. An ItemImageCre message type record is also inserted on the item_mfqueue table.

3 **Messages**: ItemImageDesc message type is created. It is a flat, synchronous message containing this item/image record. It is published as part of the item creation message when the item is approved, or flat after the original publication of the item creation.

Modify Item/Image

1 **Prerequisites**: The item/image combination already exists.

2 **Activity Detail**: The item/image record is updated in the item_image table. An ItemImageMod message type record is inserted on the item_mfqueue table.

3 **Messages**: ItemImageDesc message type is created. It is a flat, synchronous message containing this item/image record. If records that have an ItemCre and an ItemImageCre message type exist on the item_mfqueue table for this item/image, the message is updated with the "modify" message and published as part of the item creation message when the item is approved. Otherwise, it is published as a flat message.

Delete Item/Image

1 **Prerequisites**: The item/image combination already exists.

2 **Activity Detail**: The item/image record is deleted from the item_image table. An ItemImageDel message type record is inserted on the item_mfqueue table.

3 **Messages**: ItemImageRef message type is created. It is a flat, synchronous message containing the keys for this item/image record. If records that have an ItemCre and an ItemImageCre message type exist on the item_mfqueue table for this item/image, the ItemImageCre and any ItemImageMod records are deleted from the item_mfqueue table. Otherwise, it is published as a flat message.

Create Item/UDA/FreeFormat

1 **Prerequisites**: The item already exists.

2 **Activity Detail**: The item/uda/freeformat combination is inserted into the uda_item_ff table. An ItemUDAFFCre message type record is also inserted on the item_mfqueue table.

3 **Messages**: ItemUDAFFDesc message type is created. It is a flat, synchronous message containing this item/uda/freeformat record. It is published as part of the item creation message when the item is approved, or flat after the original publication of the item creation.

Modify Item/UDA/FreeFormat

1 **Prerequisites**: The item/uda/freeformat combination already exists.

2 **Activity Detail**: The item/uda/freeformat record is updated in the uda_item_ff table. An ItemUDAFFMod message type record is inserted on the item_mfqueue table.

3 **Messages**: ItemUDAFFDesc message type is created. It is a flat, synchronous message containing this item/uda/freeformat record. If records that have an ItemCre and an ItemUDAFFCre message type exist on the item_mfqueue table for this item/uda/freeformat, the message is updated with the "modify" message and published as part of the item creation message when the item is approved. Otherwise, it is published as a flat message.

Delete Item/UDA/FreeFormat

1 **Prerequisites**: The item/uda/freeformat combination already exists.

2 **Activity Detail**: The item/uda/freeformat record is deleted from the uda_item_ff table. An ItemUDAFFDel message type record is inserted on the item_mfqueue table.

3 **Messages**: ItemUDAFFRef message type is created. It is a flat, synchronous message containing the keys for this item/uda/freeformat record. If records that have an ItemCre and an ItemUDAFFCre message type exist on the item_mfqueue table for this item/uda/freeformat, the ItemUDAFFCre and any ItemUDAFFMod records are deleted from the item_mfqueue table. Otherwise, it is published as a flat message.

Create Item/UDA/LOV

1 **Prerequisites**: The item already exists.

2 **Activity Detail**: The item/uda/lov combination is inserted into the uda_item_lov table. An ItemUDALOVCre message type record is also inserted on the item_mfqueue table.

3 **Messages**: ItemUDALOVDesc message type is created. It is a flat, synchronous message containing this item/uda/lov record. It is published as part of the item creation message when the item is approved, or flat after the original publication of the item creation.

Modify Item/UDA/LOV

1 **Prerequisites**: The item/uda/lov combination already exists.

2 **Activity Detail**: The item/uda/lov record is updated in the uda_item_lov table. An ItemUDALOVMod message type record is inserted on the item_mfqueue table.

3 **Messages**: ItemUDALOVDesc message type is created. It is a flat, synchronous message containing this item/uda/lov record. If records that have an ItemCre and an ItemUDALOVCre message type exist on the item_mfqueue table for this item/uda/lov, the message is updated with the "modify" message and published as part of the item creation message when the item is approved. Otherwise, it is published as a flat message.

Delete Item/UDA/LOV

1 **Prerequisites**: The item/uda/lov combination already exists.

2 **Activity Detail**: The item/uda/lov record is deleted from the uda_item_lov table. An ItemUDALOVDel message type record is inserted on the item_mfqueue table.

3 **Messages**: ItemUDALOVRef message type is created. It is a flat, synchronous message containing the keys for this item/uda/lov record. If records that have an ItemCre and an ItemUDALOVCre message type exist on the item_mfqueue table for this item/uda/lov, the ItemUDALOVCre and any ItemUDALOVMod records are deleted from the item_mfqueue table. Otherwise, it is published as a flat message.

Create Item/UDA/Date

1 **Prerequisites**: The item already exists.

2 **Activity Detail**: The item/uda/date combination is inserted into the uda_item_date table. An ItemUDADateCre message type record is also inserted on the item_mfqueue table.

3 **Messages**: ItemUDADateDesc message type is created. It is a flat, synchronous message containing this item/uda/date record. It is published as part of the item creation message when the item is approved, or flat after the original publication of the item creation.

Modify Item/UDA/Date

1 **Prerequisites**: The item/uda/date combination already exists.

2 **Activity Detail**: The item/uda/date record is updated in the uda_item_date table. An ItemUDADateMod message type record is inserted on the item_mfqueue table.

3 **Messages**: ItemUDADateDesc message type is created. It is a flat, synchronous message containing this item/uda/date record. If records that have an ItemCre and an ItemUDADateCre message type exist on the item_mfqueue table for this item/uda/lov, the message is updated with the "modify" message and published as part of the item creation message when the item is approved. Otherwise, it is published as a flat message.

Delete Item/UDA/Date

1 **Prerequisites**: The item/uda/date combination already exists.

2 **Activity Detail**: The item/uda/date record is deleted from the uda_item_lov table. An ItemUDADateDel message type record is inserted on the item_mfqueue table.

3 **Messages**: ItemUDADateRef message type is created. It is a flat, synchronous message containing the keys for this item/uda/date record. If records that have an ItemCre and an ItemUDADateCre message type exist on the item_mfqueue table for this item/uda/date, the ItemUDADateCre and any ItemUDADateMod records are deleted from the item_mfqueue table. Otherwise, it is published as a flat message.

Delete an Item

1 **Prerequisites**: The item exists. An 'A'(Approved) item can be deleted when the user presses the "Cancel" button in the RMS dialogue after creating and approving the item.

2 **Activity Detail**: The item record is deleted from the item_master table and any child records that exist are deleted from the child tables. An ItemDel message type record is inserted on the item_mfqueue table.

3 **Message:** ItemRef message type is created. It is a flat, synchronous message containing the key for this item record. If a record that has an ItemCre message type exists on the item_mfqueue table for this item, all records for this item are deleted from the item_mfqueue table. Otherwise, it is published as a flat message.

Triggers

Trigger Description (EC_TABLE_IEM_AIUDR): This trigger fires on any insert, update or delete on the item_master table. It captures the data in the "new" bind variables for inserts and updates. It captures the "old" data on deletes. It sets the action type and message type and calls the ITEM_XML.BUILD_MESSAGE procedure to build the message. The record is inserted into the ITEM_MFQUEUE table by calling the RMSMFM_ITEMS.ADDTOQ procedure.

On Insert:

For transaction level items, sets action_type to 'A'dd and message_type to 'ItemHdrCre'.

For reference level items (below the transaction level), sets action_type to 'A'dd and message_type to 'ItemUPCCre'.

Parent and grandparent items are not published.

On Update:

For transaction level items, sets action_type to 'M'odify and message_type to 'ItemHdrMod'.

For reference level items (below the transaction level), sets action_type to 'M'odify and message_type to 'ItemUPCMod'.

On Delete:

For transaction level items, sends only the item column value for the message.

For reference level items (below the transaction level), sends only the item and item_parent column values for the message.

For transaction level items, sets action_type to 'D'elete and message_type to 'ItemHdrDel'.

For reference level items (below the transaction level), sets action_type to 'D'elete and message_type to 'ItemUPCDel'.

ITEM_XML.BUILD_MESSAGE(O_status, O_text, O_message, I_record, I_action_type)– This function is called by the trigger EC_TABLE_ITEM_AIUDR on insert, update and delete of the item_master table. This function gathers all the data necessary to build the message that needs to be sent to the Retek Integration Bus. It determines the proper message to build based on the action_type and the item type (transactional or reference) that is set in the trigger. For transaction level items, it builds ItemRef xml messages for delete statements, or ItemDesc xml messages for updates or inserts. For reference items, it builds ItemUPCRef xml messages for delete statements, or ItemUPCDesc xml messages for updates or inserts.

Trigger Description (EC_TABLE_ISP_AIUDR): This trigger fires on any insert, update or delete on the item_supplier table. It captures the data in the “new” bind variables for inserts and updates. It captures the “old” data on deletes. It sets the action type and message type and calls the ITEMSUPPLIER_XML.BUILD_MESSAGE procedure to build the message. The record is inserted into the ITEM_MFQUEUE table by calling the RMSMFM_ITEMS.ADDTOQ procedure.

<u>On Insert:</u>

Sets action_type to ‘A’dd and message_type to ‘ItemSupCre’.

<u>On Update</u>:

Sets action_type to ‘M’odify and message_type to ‘ItemSupMod’.

<u>On Delete:</u>

Sends only the item and supplier column values for the message.

Sets action_type to ‘D’elete and message_type to ‘ItemSupDel’.

<u>ITEMSUPPLIER_XML.BUILD_MESSAGE</u>(O_status, O_text, O_message, I_record, I_action_type)– This function is called by the trigger EC_TABLE_ISP_AIUDR on insert, update and delete of the item_supplier table. This function gathers all the data necessary to build the message that needs to be sent to the Retek Integration Bus. It determines the proper message to build based on the action_type that is set in the trigger. It builds ItemSupRef xml messages for delete statements, or ItemSupDesc xml messages for updates or inserts.

Trigger Description (EC_TABLE_ISC_AIUDR): This trigger fires on any insert, update or delete on the item_supp_country table. It captures the data in the “new” bind variables for inserts and updates. It captures the “old” data on deletes. It sets the action type and message type and calls the ISC_XML.BUILD_MESSAGE procedure to build the message. The record is inserted into the ITEM_MFQUEUE table by calling the RMSMFM_ITEMS.ADDTOQ procedure.

<u>On Insert:</u>

Sets action_type to ‘A’dd and message_type to ‘ItemSupCtyCre’.

<u>On Update</u>:

Sets action_type to ‘M’odify and message_type to ‘ItemSupCtyMod’.

<u>On Delete:</u>

Sends only the item, supplier and origin_country_id column values for the message.

Sets action_type to ‘D’elete and message_type to ‘ItemSupCtyDel’.

ISC_XML.BUILD_MESSAGE(O_status, O_text, O_message, I_record, I_action_type)– This function is called by the trigger EC_TABLE_ISC_AIUDR on insert, update and delete of the item_supp_country table. This function gathers all the data necessary to build the message that needs to be sent to the Retek Integration Bus. It determines the proper message to build based on the action_type that is set in the trigger. It builds ItemSupCtyRef xml messages for delete statements, or ItemSupCtyDesc xml messages for updates or inserts.

Trigger Description (EC_TABLE_ISD_AIUDR): This trigger fires on any insert, update or delete on the item_supp_country_dim table. It captures the data in the "new" bind variables for inserts and updates. It captures the "old" data on deletes. It sets the action type and message type and calls the ISCD_XML.BUILD_MESSAGE procedure to build the message. The record is inserted into the ITEM_MFQUEUE table by calling the RMSMFM_ITEMS.ADDTOQ procedure.

On Insert:

Sets action_type to 'A'dd and message_type to 'ISCDimCre'.

On Update:

Sets action_type to 'M'odify and message_type to 'ISCDimMod'.

On Delete:

Sends only the item, supplier, origin_country_id and dim_object column values for the message.

Sets action_type to 'D'elete and message_type to 'ISCDimDel'.

ISC_XML.BUILD_MESSAGE(O_status, O_text, O_message, I_record, I_action_type)– This function is called by the trigger EC_TABLE_ISC_AIUDR on insert, update and delete of the item_supp_country_dim table. This function gathers all the data necessary to build the message that needs to be sent to the Retek Integration Bus. It determines the proper message to build based on the action_type that is set in the trigger. It builds ISCDimRef xml messages for delete statements, or ISCDimDesc xml messages for updates or inserts.

Trigger Description (EC_TABLE_PKS_AIUDR: This trigger fires on any insert, update or delete on the packitem_breakout table. It captures the data in the "new" bind variables for inserts and updates. It captures the "old" data on deletes. It populates a PL/SQL table with this data.

Trigger Description (EC_TABLE_PKS_IUDS: This trigger fires on any insert, update or delete on the packitem_breakout table. It loops through the PL/SQL table that was populated in the row trigger and determines the value for the packitem quantity in the message based on what is retrieved from the v_packsku_qty view and the DML event. It calls the **ITEMBOM_XML.BUILD_MESSAGE** procedure to build the message. The record is inserted into the ITEM_MFQUEUE table by calling the RMSMFM_ITEMS.ADDTOQ procedure.

On Insert:

If the v_packsku_qty quantity is equal to the record just added, it sets action_type to 'A'dd and message_type to 'ItemBOMCre'. If not, it sets action_type to 'M'odify and message_type to 'ItemBOMMod'.

On Update:

Sets action_type to 'M'odify and message_type to 'ItemBOMMod'.

On Delete:

Sends only the pack_no and item column values for the message.

If the packitem quantity is 0, it sets action_type to 'D'elete and message_type to 'ItemBOMDel'.

If the absolute value of packitem quantity is greater than 0, it sets action_type to 'M'odify and message_type to 'ItemBOMMod'.

ITEMBOM_XML.BUILD_MESSAGE(O_status, O_text, O_message, I_record, I_action_type)– This function is called by the trigger EC_TABLE_PKS_IUDS on insert, update and delete of the item_supplier table. This function gathers all the data necessary to build the message that needs to be sent to the Retek Integration Bus. It determines the proper message to build based on the action_type that is set in the trigger. It builds ItemBOMRef xml messages for delete statements, or ItemBOMDesc xml messages for updates or inserts.

Trigger Description (EC_TABLE_IIM_AIUDR): This trigger fires on any insert, update or delete on the item_image table. It captures the data in the "new" bind variables for inserts and updates. It captures the "old" data on deletes. It sets the action type and message type and calls the ITEMIMAGE_XML.BUILD_MESSAGE procedure to build the message. The record is inserted into the ITEM_MFQUEUE table by calling the RMSMFM_ITEMS.ADDTOQ procedure.

On Insert:

Sets action_type to 'A'dd and message_type to 'ItemImageCre'.

On Update:

Sets action_type to 'M'odify and message_type to 'ItemImageMod'.

On Delete:

Sends only the item and image_name column values for the message.

Sets action_type to 'D'elete and message_type to 'ItemImageDel'.

ITEMIMAGE_XML.BUILD_MESSAGE(O_status, O_text, O_message, I_record, I_action_type)– This function is called by the trigger EC_TABLE_IIM_AIUDR on insert, update and delete of the item_image table. This function gathers all the data necessary to build the message that needs to be sent to the Retek Integration Bus. It determines the proper message to build based on the action_type that is set in the trigger. It builds ItemImageRef xml messages for delete statements, or ItemImageDesc xml messages for updates or inserts.

Trigger Description (EC_TABLE_UIF_AIUDR): This trigger fires on any insert, update or delete on the uda_item_ff table. It captures the data in the "new" bind variables for inserts and updates. It captures the "old" data on deletes. It sets the action type and message type and calls the UDA_ITEM_XML.BUILD_MESSAGE procedure to build the message. The record is inserted into the ITEM_MFQUEUE table by calling the RMSMFM_ITEMS.ADDTOQ procedure.

<u>On Insert:</u>

Sets action_type to 'A'dd and message_type to 'ItemUDAFFCre'.

<u>On Update</u>:

Sets action_type to 'M'odify and message_type to 'ItemUDAFFMod'.

<u>On Delete:</u>

Sends only the item and uda_id column values for the message.

Sets action_type to 'D'elete and message_type to 'ItemUDAFFDel'.

<u>UDA_ITEM_XML.BUILD_MESSAGE</u>(O_status, O_text, O_message, I_record, I_action_type)– This function is called by the trigger EC_TABLE_UIF_AIUDR on insert, update and delete of the item_supplier table. This function gathers all the data necessary to build the message that needs to be sent to the Retek Integration Bus. It determines the proper message to build based on the action_type that is set in the trigger. It builds ItemUDAFFRef xml messages for delete statements, or ItemUDAFFDesc xml messages for updates or inserts.

Trigger Description (EC_TABLE_UIL_AIUDR): This trigger fires on any insert, update or delete on the uda_item_lov table. It captures the data in the "new" bind variables for inserts and updates. It captures the "old" data on deletes. It sets the action type and message type and calls the UDA_ITEM_XML.BUILD_MESSAGE procedure to build the message. The record is inserted into the ITEM_MFQUEUE table by calling the RMSMFM_ITEMS.ADDTOQ procedure.

<u>On Insert:</u>

Sets action_type to 'A'dd and message_type to 'ItemUDALOVCre'.

<u>On Update</u>:

Sets action_type to 'M'odify and message_type to 'ItemUDALOVMod'.

<u>On Delete:</u>

Sends only the item, uda_id and uda_value column values for the message.

Sets action_type to 'D'elete and message_type to 'ItemUDALOVDel'.

UDA_ITEM_XML.BUILD_MESSAGE(O_status, O_text, O_message, I_record, I_action_type)– This function is called by the trigger EC_TABLE_UIL_AIUDR on insert, update and delete of the item_supplier table. This function gathers all the data necessary to build the message that needs to be sent to the Retek Integration Bus. It determines the proper message to build based on the action_type that is set in the trigger. It builds ItemUDALOVRef xml messages for delete statements, or ItemUDALOVDesc xml messages for updates or inserts.

Trigger Description (EC_TABLE_UID_AIUDR): This trigger fires on any insert, update or delete on the uda_item_date table. It captures the data in the "new" bind variables for inserts and updates. It captures the "old" data on deletes. It sets the action type and message type and calls the UDA_ITEM_XML.BUILD_MESSAGE procedure to build the message. The record is inserted into the ITEM_MFQUEUE table by calling the RMSMFM_ITEMS.ADDTOQ procedure.

On Insert:

Sets action_type to 'A'dd and message_type to 'ItemUDADateCre'.

On Update:

Sets action_type to 'M'odify and message_type to 'ItemUDADateMod'.

On Delete:

Sends only the item and uda_id column values for the message.

Sets action_type to 'D'elete and message_type to 'ItemUDADateDel'.

UDA_ITEM_XML.BUILD_MESSAGE(O_status, O_text, O_message, I_record, I_action_type)– This function is called by the trigger EC_TABLE_UID_AIUDR on insert, update and delete of the item_supplier table. This function gathers all the data necessary to build the message that needs to be sent to the Retek Integration Bus. It determines the proper message to build based on the action_type that is set in the trigger. It builds ItemUDADateRef xml messages for delete statements, or ItemUDADateDesc xml messages for updates or inserts.

Message Family Manager Procedures

Public Procedures:

ADDTOQ(O_status, O_text, I_queue_rec, I_message) – This procedure is called by a DML event capture trigger, and takes the message type, family key values and, for synchronously captured messages, the message itself.

First it checks the input parameter for the item status that is part of the I_queue_rec input record. This input record is defined the package specification. If the item status is 'A'(approved), it sets a local variable to 'Y'(yes) and uses this local variable as the value for the approve_ind column in the insert statement. It inserts a record into the item_mfqueue table using the sequence for the table, the values from the input record parameter, the local variable for the approve_ind and the input CLOB parameter that contains the data in XML format.

GETNXT (O_status_code, O_error_msg, O_message_type, O_message, O_item, O_supplier, O_country_id, O_dim_object, O_upc, O_bom_comp, O_image_name, O_uda_id, O_uda_value, O_sellable_ind, I_num_threads, I_thread_val)– This publicly exposed procedure is typically called by a RIB publication adaptor. Its parameters are well defined and arranged in a specific order. The message type is the RIB defined short message name, the message is the xml message, and the family key(s) are the key for the message as pertains to the family, not all of which will necessarily be populated for all message types.

The procedure produces a message through the following steps:

It loops through the item_mfqueue table records that have a pub_status of 'U'(Unpublished).

If the return from the CREATE_PREVIOUS function is TRUE

- calls the CLEAN_QUEUE procedure.
- if the approve_ind column equals 'Y'(Yes)
  - calls the MAKE_CREATE procedure
  - assigns all the output parameters with the values from the current item_mfqueue row except for O_message which is returned from the MAKE_CREATE procedure and sets O_status to API_CODES.SUCCESS.
- if the CAN_CREATE returns FALSE, sets the pub_status field of the current item_mfqueue row to 'N' and updates the row.
- If the return from the CREATE_PREVIOUS function is FALSE
- assigns all the output parameters with the values from the current item_mfqueue row and set O_status to API_CODES.SUCCESS.
- call the DELETE_QUEUE_REC to delete the row from the item_mfqueue table.

If no "publishable" messages are retrieved from the above steps the procedure returns a status of 'N'(No message).

Status code is one of five values, as shown in the following table. For more discussion of the status codes, refer to the Error Handling Guidelines document or the process flow in the following section. These codes come from an EAI team defined RIB_CODES package.

Private Procedures:

These private procedures are only necessary when the initial create message is hierarchical. If all messages in the family are flat, there is no need for these procedures.

CREATE_PREVIOUS(O_status, O_text, I_queue_rec) – This function determines if a header level create already exists on the queue table for the same key value and with a sequence number less than the current records sequence number.

It checks the item_mfqueue table for the existence of a row for that has an item equal to the passed in value for item, a message type equal to the value of ItemCre and a seq_no that is less than or equal to the passed in value for seq_no. If such a row exists in the table, it returns TRUE.

CLEAN_QUEUE(O_status, O_text, I_queue_rec) – This procedure cleans up the queue by eliminating modification messages. It is only called if CREATE_PREVIOUS returns true. For each modification message type, it finds the previous corresponding create message type. It then calls REPLACE_QUEUE to copy the modify message into the create message and calls DELETE_QUEUE_REC to delete the modify record. For each delete message type, it finds the previous corresponding messages. It then calls DELETE_QUEUE_REC to delete the create message record.

The following examples illustrate the flow of the logic in this procedure for the item family:

First it checks the message_type passed to procedure for the value of any of the item delete message types, i.e. ItemSupCtyDel, ItemUPCDel, etc. If the message_type is a "delete" message, it deletes records from the item_mfqueue table for the appropriate key values and for the seq_no less than or equal to the passed in value for seq_no.

Example for the message type ItemSupCtyDel:

```
delete from item_mfqueue
 where supplier = I_queue_rec.supplier
   and item     = I_queue_rec.item
   and country  = I_queue_rec.country
   and seq_no  <= I_queue_rec.seq_no;
```

If the message_type is an “update” message such as ItemSupMod, it assigns the corresponding “Add” message_type to a local variable.

Example for the message type ISCDimMod:

```
L_create_type := ISCDimAdd;
```

If this local variable is not null and if the call to REPLACE_QUEUE returns TRUE, it calls DELETE_QUEUE_REC to delete the row from item_mfqueue.

REPLACE_QUEUE(O_status, O_text, I_rec, I_message_type) – This procedure replaces the message in the “create” message type record with the message from a “Modify” message type record.

It locks the item_mfqueue table for all rows that have a seq_no less than the passed in value for seq_no. It updates the message column with the passed in value for message for the row that matches the key values passed in the record to the function and that matches the message_type passed as a parameter. It uses the nvl function for all key columns except item because these key values are optional and dependent on the message_type.

DELETE_QUEUE_REC(O_status, O_text, I_seq_no) – This procedure deletes from the item_mfqueue table the row that has the seq_no column value equal to the sequence value passed to the procedure.

MAKE_CREATE(O_status, O_text, O_msg, I_queue_rec) – This procedure combines the current message and all previous messages with the same key in the queue table to create the complete hierarchical message. It first copies the header clob into a local variable. It then creates a new message clob and appends the clob in the local variable to the new clob. The remainder of this procedure gets each of the details grouped by their document type and adds them to the new message. When it is finished creating the new message, it deletes all the records from the queue for that item with a sequence number less than or equal to the current records sequence number. This new message is passed back to the bus.

For the Item this procedure is implemented as follows:

Two cursors are used. One cursor cursor retrieves the row from the item_mfqueue table for the item_master message, the item is equal to the value of the passed in item, the seq_no is less than the passed in seq_no and the message_type is equal to ‘ItemCre’. The other cursor retrieves all the item related messages for the item details, the item is equal to the value of the passed in item, the seq_no is less than or equal to the sequence value passed to the procedure and the message_type is equal message_type value passed to the cursor. Order the second cursor by seq_no. A local procedure with parameters for message_type and message_name, adds the detail message to the header message. It loops through the second cursor with the value of the message type parameter and do the following:

- It appends the message read in to the new message being created.

It adds all the item related detail messages by calling the local procedure described above for each item detail and passing the message type and the message name unique to the item detail. It uses the constants define in the package spec for these values. The order for adding an item detail to the XML message is specified in the item DTD.

Finally, it closes the new clob message and deletes all rows that comprise this message from the item_mfqueue table.

Design Assumptions

- One of the primary assumptions in the current 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.

Outstanding Technical Issues

- Seebeyond can only handle 30 character procedure names. So, the entire name of the call, package.procedure has to be within 30 characters. This only applies to public procedures, not internal, private functions and procedures.
- Seebeyond insists, for some unknown reason, on unique parameter names for functions. Therefore, each O_status must be unique for the public procedures thus the AR, GR, etc, appending the parameter names. This only applies to public procedures, not internal, private functions and procedures. Also Seebeyond is expecting to map to certain output parameter names in the GETNXT function.

Currency Exchange Rates Subscription Design

Functional Area

RMS subscribing to Currency Exchange Rates.

Design Overview

Data Flow

An external system will publish a currency exchange rate, thereby placing the currency exchange rate information onto the RIB (Retek Information Bus). RMS will subscribe to the currency exchange rate 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 currency exchange rate message is a flat message that will consist of a currency exchange rate record.

The record will contain information about the currency exchange rate as a whole.

Subscription Procedures

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

Public API Procedures:

RMSSUB_CURRATECRE.CONSUME (O_STATUS_CODE, O_ERROR_MESSAGE, I_MESSAGE) - 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 currency exchange rate message consisting of the aforementioned record. The procedure will then place a call to the main RMSSUB_CURRXRATE.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 currency exchange rate table depending upon the success of the validation.

Private Internal Functions and Procedures (rmssub_curratecre.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, IO_text, I_cause, I_program)- 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_CURRXRATE package and all errors that occur during subscription in the RMSSUB_CURRATECRE 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_CURRXRATE.

Main Consume Function:

RMSSUB_CURRXRATE.CONSUME (O_ERROR_MESSAGE, I_MESSAGE) - This procedure accepts a XML file in the form of an Oracle CLOB data type from the RIB (I_message) from the aforementioned public curratecre procedure whenever a message is made available by the RIB. This message will consist of the aforementioned record.

The procedure will then 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 appropriate currency exchange rate database table depending upon the success of the validation.

XML Parsing:

PARSE_HEADER(O_ERROR_MESSAGE, O_CURR_RECORD, I_CURR_ROOT) – This function will used to extract the currency exchange rate level information from the Currency Exchange Rate XML file and place that information onto an internal Currency Exchange Rate record.

Record is based upon the record type curr_rectype.

Validation:

PROCESS_HEADER(O_ERROR_MESSAGE, I_CURR_RECORD) – After the values are parsed for a particular currency exchange rate record, RMSSUB_CURRXRATE.CONSUME will call this function, which will in turn call various functions inside RMSSUB_CURRXRATE in order to validate the values and place them on the appropriate currency exchange rate table depending upon the success of the validation. PROCESS_TERMS is called to actually insert or update the currency exchange rate table.

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.

Outstanding Technical Issues

Seebeyond can only handle 30 character procedure names. So, the entire name of the call, package.procedure has to be within 30 characters. This only applies to public procedures, not internal, private functions and procedures.

Freight Terms Subscription Design

Functional Area

RMS subscribing to Freight Terms.

Design Overview

Data Flow

An external system will publish a freight term, thereby placing the freight term information onto the RIB (Retek Information Bus). RMS will subscribe to the freight term 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 freight term message is a flat message that will consist of a freight term record.

The record will contain information about the freight term as a whole.

Subscription Procedures

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

Public API Procedures

RMSSUB_FRTTERMCRE.CONSUME (O_STATUS_CODE, O_ERROR_MESSAGE, I_MESSAGE) - 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 freight term message consisting of the aforementioned record. The procedure will then place a call to the main RMSSUB_FTERM.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 freight term table depending upon the success of the validation.

Private Internal Functions and Procedures (rmssub_frttermcre.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, IO_text, I_cause, I_program)- 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_FTERM package and all errors that occur during subscription in the RMSSUB_FRTTERMCRE 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_FTERM:

Main Consume Function

RMSSUB_FTERM.CONSUME (O_STATUS_CODE, O_ERROR_MESSAGE, I_MESSAGE) - This procedure accepts a XML file in the form of an Oracle CLOB data type from the RIB (I_message) from the aforementioned public frttermcre procedure whenever a message is made available by the RIB. This message will consist of the aforementioned record.

The procedure will then 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 appropriate freight term database table depending upon the success of the validation.

XML Parsing:

PARSE_FTERM (O_ERROR_MESSAGE, O_FTERM_RECORD, I_FTERM_ROOT) – This function will used to extract the freight term level information from the Freight Term XML file and place that information onto an internal Freight Term record.

Record is based upon the fterm_sql package record: fterm_record.

Validation:

PROCESS_FTERM(O_ERROR_MESSAGE, I_FTERM_RECORD) – After the values are parsed for a particular freight term record, RMSSUB_FTERM.CONSUME will call this function, which will in turn call various functions inside RMSSUB_FTERM in order to validate the values and place them on the appropriate freight_terms table depending upon the success of the validation. FTERM_SQL.PROCESS_TERMS is called to actually insert or update the freight terms table.

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.

Outstanding Technical Issues

Seebeyond can only handle 30 character procedure names. So, the entire name of the call, package.procedure has to be within 30 characters. This only applies to public procedures, not internal, private functions and procedures.

GL Chart of Accounts Subscription Design

Functional Area

General ledger chart of accounts

Design Overview

Data Flow

An external system will publish GL Chart of Accounts, thereby placing the GL chart of accounts information onto the RIB (Retek Information Bus). RMS will subscribe to the GL chart of accounts 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 GL chart of accounts message is a flat message that will consist of a GL chart of accounts record.

The record will contain information about the GL chart of accounts as a whole.

Subscription Procedures

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

Public API Procedures

RMSSUB_GLCOACRE.CONSUME (O_STATUS_CODE, O_ERROR_MESSAGE, I_MESSAGE) - 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 GL chart of accounts message consisting of the aforementioned record. The procedure will then place a call to the main RMSSUB_GLCACCT.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 gl chart of accounts table depending upon the success of the validation.

Private Internal Functions and Procedures (rmssub_glcoacre.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, IO_text, I_cause, I_program)- 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_GLCACCT package and all errors that occur during subscription in the RMSSUB_GLCOACRE 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_GLCACCT.

Main Consume Function

RMSSUB_GLCACCT.CONSUME (O_ERROR_MESSAGE, I_MESSAGE) - This procedure accepts a XML file in the form of an Oracle CLOB data type from the RIB (I_message) from the aforementioned public glcoa procedure whenever a message is made available by the RIB. This message will consist of the aforementioned record.

The procedure will then 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 appropriate gl chart of accounts database table depending upon the success of the validation.

XML Parsing:

PARSE_HEADER(O_ERROR_MESSAGE, O_GLACCT_RECORD, I_GLACCT_ROOT) – This function will used to extract the GL chart of accounts level information from the GL Chart of Accounts XML file and place that information onto an internal GL Chart of Accounts record.

Record is based upon the record type glacct_rectype.

Validation:

PROCESS_HEADER(O_ERROR_MESSAGE, I_GLACCT_RECORD) – After the values are parsed for a particular GL chart of accounts record, RMSSUB_GLCACCT.CONSUME will call this function, which will in turn call various functions inside RMSSUB_GLCACCT in order to validate the values and place them on the appropriate GL chart of accounts table depending upon the success of the validation. PROCESS_GLACCT is called to actually insert or update the GL chart of accounts table.

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 straightforward manner.
- The adaptor is only setup to call stored procedures, not stored functions. Any public program then needs to be a procedure.

Outstanding Technical Issues

Seebeyond can only handle 30 character procedure names. So, the entire name of the call, package.procedure has to be within 30 characters. This only applies to public procedures, not internal, private functions and procedures.

Payment Terms Subscription Design

Functional Area

RMS subscribing to Payment Terms.

Design Overview

Data Flow

An external system will publish a payment term, thereby placing the payment term information onto the RIB (Retek Information Bus). RMS will subscribe to the payment term 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 payment term message is a flat message that will consist of a payment term record.

The record will contain information about the payment term as a whole.

Subscription Procedures

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

Public API Procedures:

RMSSUB_PAYTERMCRE.CONSUME (O_STATUS_CODE, O_ERROR_MESSAGE, I_MESSAGE) - 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 payment term message consisting of the aforementioned record. The procedure will then place a call to the main RMSSUB_PTRM.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 payment term table depending upon the success of the validation.

Private Internal Functions and Procedures (rmssub_paytermcre.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, IO_text, I_cause, I_program)- 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_PTRM package and all errors that occur during subscription in the RMSSUB_PAYTERMCRE 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_PTRM.

Main Consume Function:

RMSSUB_PTRM.CONSUME (O_ERROR_MESSAGE, I_MESSAGE) - This procedure accepts a XML file in the form of an Oracle CLOB data type from the RIB (I_message) from the aforementioned public paytermcre procedure whenever a message is made available by the RIB. This message will consist of the aforementioned record.

The procedure will then 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 appropriate payment term database table depending upon the success of the validation.

XML Parsing:

PARSE_HEADER(O_ERROR_MESSAGE, O_TERM_RECORD, I_TERM_ROOT) – This function will used to extract the payment term level information from the Payment Term XML file and place that information onto an internal Payment Term record.

Record is based upon the record type pay_rectype.

Validation:

PROCESS_HEADER(O_ERROR_MESSAGE, I_TERM_RECORD) – After the values are parsed for a particular payment term record, RMSSUB_PTRM.CONSUME will call this function, which will in turn call various functions inside RMSSUB_PTRM in order to validate the values and place them on the appropriate terms table depending upon the success of the validation. PROCESS_TERMS is called to actually insert or update the payment terms table.

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 straightforward manner.
- The adaptor is only setup to call stored procedures, not stored functions. Any public program then needs to be a procedure.

Outstanding Technical Issues

Seebeyond can only handle 30 character procedure names. So, the entire name of the call, package.procedure has to be within 30 characters. This only applies to public procedures, not internal, private functions and procedures.

Supplier Subscription Design

Functional Area

RMS Subscribing to Supplier data.

Design 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.

Subscription Procedures

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).

Public API Procedures:

RMSSUB_VENDORCRE.CONSUME (O_STATUS_CODE, O_ERROR_MESSAGE, I_MESSAGE) - 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, IO_text, I_cause, I_program)- 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_CODE, O_ERROR_MESSAGE, I_MESSAGE) - 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 then 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 appropriate supplier and address database tables depending upon the success of the validation.

XML Parsing:

PARSE_SUPPLIER (O_ERROR_MESSAGE, O_TABLE_LOCKED, O_SUPPLIER_RECORD, I_SUPPLIER_ROOT) – 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, O_TABLE_LOCKED, O_ADDRESS_RECORD, I_ADDR_NODE) – 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, O_TABLE_LOCKED, IO_SUPPLIER_RECORD) – 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, O_TABLE_LOCKED, I_SUPPLIER_NO, I_ADDRESS_RECORD) – After the values are parsed for a particular address 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 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.

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 straightforward manner.
- The adaptor is only setup to call stored procedures, not stored functions. Any public program then needs to be a procedure.

Outstanding Technical Issues

Seebeyond can only handle 30 character procedure names. So, the entire name of the call, package.procedure has to be within 30 characters. This only applies to public procedures, not internal, private functions and procedures.

Volume 4 – Batch designs

The following batch program designs have been updated for RMS 10.1 and are included in this section:

- DEALUPLD
- DITINSRT
- EDIDLCON
- EDIDLORD
- EDIUPCAT
- ONORDEXT
- POSDNLD
- POSUPLD
- PRECOSTCALC
- PRMPCUPD
- SOUTDNLD
- TCKTDNLD
- VATDLXPL
- WASTEADJ

Deal upload [dealupld]

Design Overview

This program will read a Retek flat file, which was created by an external program and will upload information from this file to the following Deals Management tables in RMS: DEAL_HEAD, DEAL_DETAIL, DEAL_THRESHOLD, DEAL_ITEMLOC, POP_TERMS_DEF.

The external program will translate the standard EDI upload file into the RMS format specified below and will also translate the content of the EDI input file into RMS values (that is: a location's DUNS number to its RMS number) using the RMS database.

Dealupld.pc will take the above output file and use it as an input file to upload data into the Deals Management RMS database. The LUW is a single Deal Head Detail record and its associated component records in the input file. Therefore in each loop of the program one deal is uploaded at a time.

The program will verify that check constraints are not violated before the inserts actually take place. If a validation fails, a warning is written to the batch error file and the program resumes processing. Once the deal is processed, a check is made whether any warnings occurred. If yes, no insert occurs to the database but the entire deal is written to the reject file. This allows the user to see not only the first validation failure, but all of them before the deal was written out to the reject file.

See Section XIII Design Assumptions for more information on how records in the input file should be formatted and looped.

Stored Procedures / Shared Modules (Maintainability)

DEAL_SEQUENCE – This sequence is used to get the next deal_id for the deal being uploaded.

Input Specifications

'File-To-Table'

All fields that are of type Char in the input file format description above should be left justified and padded with the space character. If the field is not required and no value is being uploaded, the entire field should be padded with the space character.

Date fields are always formatted 'YYYYMMDDHH24MISS' (Char(14)).

All fields that are of type Number in the input file format description above should be right justified and padded with 0s. If the field is not required and no value is being uploaded, the entire field should be padded with 0s. All Number fields which have an implied decimal (for example: Number(10,4), but not Number(8)) are in effect the length of the number PLUS its decimal places. (for example: 14 byte wide field to hold a Number(10,4).) The program will upload the value formatted to include the 4 decimal digits (for example: Number(10,4): 01234567891234 -> 123456789.1234, Number(6,2): 12345600 -> 123456.00). Therefore if a number of size Number(20,4) is uploaded, this number will take up 24 bytes in the input file.

The input file must have the following structure:

```
FHEAD
{
        THEAD of DHDTL    REQUIRED    for deal head record
             TDETL        REQUIRED    1 deal head record
        TTAIL             REQUIRED    end of deal head record
        THEAD of DCDTL    REQUIRED    for deal component records
        [
             TDETL        OPTIONAL    for deal component records
        ]
        TTAIL             REQUIRED    end of deal component records
        THEAD of DIDTL    REQUIRED    for item-loc records
        [
             TDETL        OPTIONAL    for item-loc records
        ]
        TTAIL             REQUIRED    end of item-loc records
        THEAD of PPDTL    REQUIRED    for proof of performance records
        [
             TDETL        OPTIONAL    for proof of performance records
        ]
        TTAIL             REQUIRED    end of proof of performance records
        THEAD of DTDTL    REQUIRED    for threshold records
        [
             TDETL        OPTIONAL    for threshold records
        ]
        TTAIL             REQUIRED    end of threshold records
}
FTAIL
```

The set between the curly brackets may be looped to upload multiple deals from the same file. Within each set, the TDETL records in angle brackets may be sub-looped as a sub-set of the main set.

See Section XIII Design Assumptions for more information on how records in the input file should be formatted and looped.

Input File

Record Name	Field Name	Field Type	Default Value	Description/Constraints
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type (the beginning of the input file).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	File Type Definition	Char(5)	EDIDU	Identifies file as 'EDI Deals Upload'
	File Create Date	Char(14)	Create date	current date, formatted to 'YYYYMMDDHH24MISS'.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
Transaction Header	**File Type Record Descriptor**	**Char(5)**	**THEAD**	**Identifies file record type to upload a new deal header.**
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Detail Record Type	Char(5)	DHDTL	Identifies file record type Deal Header. This record MUST BE FOLLOWED BY ONE AND ONLY ONE REQUIRED TDETL RECORD that holds the deal head information.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload a new deal.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Partner Type	Char(6)	REQUIRED	Type of the partner the deal applies to. Valid values are 'S' for a supplier, 'S1' for supplier hierarchy level 1 (e.g. manufacturer), 'S2' for supplier hierarchy level 2 (e.g. distributor) and 'S3' for supplier hierarchy level 3 (e.g. wholesaler). Descriptions of these codes will be held on the codes table under a code_type of 'SUHL'. Information pertaining to a single deal has to belong to the same supplier, since a deal may have only one supplier hierarchy associated with it. Only items with the same supplier hierarchy can be on the same deal. Supplier hierarchy is stored at an item / supplier / country / location level.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Partner Id	Char(10)	Blank (space character string)	Level of supplier hierarchy (e.g. manufacturer, distributor or wholesaler), set up as a partner in the PARTNER table, used for assigning rebates by a level other than supplier. Rebates at this level will include all eligible supplier/item/country records assigned to this supplier hierarchy level. This field is required if the Partner Type field was set to 'S1', 'S2' or 'S3'. This field must be blank if the Partner Type field was set to 'S'.
	Supplier	Number(10)	Blank (space character string)	Deal supplier's number. This supplier can be at any level of supplier hierarchy. This field is required if the Partner Type field was set to 'S'. This field must be blank if the Partner Type field was set to 'S1', 'S2' or 'S3'.
	Type	Char(6)	REQUIRED	Type of the deal. Valid values are A for annual deal, P for promotional deal, O for PO-specific deal or M for vendor-funded markdown. Deal types will be held on the codes table under a code type of 'DLHT'.
	Currency Code	Char(3)	Blank (space character string)	Currency code of the deal's currency. All costs on the deal will be held in this currency. If Type is 'O', 'P' or 'A', then Currency Code may not be blank. Currency Code has to be blank if Type is 'M'.
	Active Date	Char(14)	REQUIRED	Date the deal will become active. This date will determine when deal components begin to be factored into item costs. For a PO-specific deal, the active_date will be the order's written date.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Close Date	Char(14)	Blank (space character string)	Date the deal will/did end. This date determines when deal components are no longer factored into item costs. It is optional for annual deals, required for promotional deals. It will be left NULL for PO-specific deals. Close Date must not be blank if Type is 'P' or ‘M’. Close Date has to be blank if Type is 'O'.
	External Reference Number	Char(30)	Blank (space character string)	Any given external reference number that is associated with the deal.
	Order Number	Number(8)	Blank (space character string)	Order the deal applies to, if the deal is PO-specific.
	Recalculate Approved Orders	Char(1)	REQUIRED	Indicates if approved orders should be recalculated based on this deal once the deal is approved. Valid values are Y for yes or N for no. Valid values are ‘Y’ and ‘N’.
	Comments	Char(2000)	Blank (space character string)	Free-form comments entered with the deal.
Transacti on Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail). For DHDTL TDETL records this will always be 1!
Transacti on Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal sub loop.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Detail Record Type	Char(5)	DCDTL	Identifies file record type of sub loop as Deal Component Detail.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload deal components.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Deal Component Type	Char(6)	REQUIRED	Type of the deal component, user-defined and stored on the DEAL_COMP_TYPE table.
	Application Order	Number(10)	Blank (space character string)	Number indicating the order in which the deal component should be applied with respect to any other deal components applicable to the item within the deal. This number will be unique across all deal components within the deal. It must be NULL for an M-type deal (vendor funded markdown).
	Billing Type	Char(6)	REQUIRED	Billing type of the deal component. Valid values are 'OI' for off-invoice, 'BD' for bill-back with debit memo or 'BC' for bill-back with credit note request. Billing types will be held on the codes table under a code type of 'DLBT'.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Bill Back Period	Char(6)	Blank (space character string)	Code that identifies the bill-back period for the deal component. This field will only be populated for billing types of 'BD' or 'BC'. Bill back period codes will be user-defined and stored on the BILL_BACK_PERIOD table. If Billing Type is 'BD' or 'BC' then Bill Back Period must not be blank, otherwise it has to be blank.
	Collect Start Date	Char(14)	Blank (space character string)	Date that collection of the bill-back should begin. If Billing Type is 'BD' or 'BC' then Collect Start Date must not be blank, otherwise it has to be blank.
	Collect End Date	Char(14)	Blank (space character string)	Date that collection of the bill-back should end. If Billing Type is 'BD' or 'BC' then Collect End Date must not be blank, otherwise it has to be blank.
	Deal Application Timing	Char(6)	Blank (space character string)	Indicates when the deal component should be applied - at PO approval or time of receiving. Valid values are 'O' for PO approval, 'R' for receiving. These values will be held on the codes tables under a code type of 'AALC'. It must be NULL for an M-type deal (vendor funded markdown).
	Cost Application Level Indicator	Char(6)	Blank (space character string)	Indicates what cost bucket the deal component should affect. Valid values are 'N' for net cost, 'NN' for net net cost and 'DNN' for dead net net cost. These values will be held on the codes tables under a code type of 'DLCA'. It must be NULL for an M-type deal (vendor funded markdown).

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Pricing Cost Indicator	Char(1)	REQUIRED	Identifies deal components that should be included when calculating a pricing cost. Valid values are ‘Y’es and ‘N’o.
	Deal Class	Char(6)	Blank (space character string)	Identifies the calculation class of the deal component. Valid values are 'CU' for cumulative (discounts are added together and taken off as one lump sum), 'CS' for cascade (discounts are taken one at a time with subsequent discounts taken off the result of the previous discount) and 'EX' for exclusive (overrides all other discounts). 'EX' type deal components are only valid for promotional deals. Deal classes will be held on the codes table under a code type of 'DLCL'. It must be NULL for an M-type deal (vendor funded markdown).
	Threshold Limit Type	Char(6)	Blank (space character string)	Identifies whether thresholds will be set up as qty values, currency amount values or percentages (growth rebates only). Valid values are 'Q' for qty, 'A' for currency amount or 'P' for percentage. Threshold limit types will be held on the codes table under a code type of 'DLLT'. It must be NULL for an M-type deal (vendor funded markdown) or if the threshold value type is ‘Q’ (buy/get deals). If Growth Rebate Indicator is 'Y', then the Threshold Limit Type has to be 'P', otherwise 'Q', 'A' or NULL.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Threshold Limit Unit of Measure	Char(4)	Blank (space character string)	Unit of measure of the threshold limits, if the limit type is quantity. Only Unit of Measures with a UOM class of 'VOL' (volume), 'MASS' or 'QTY' (quantity) can be used in this field. Valid Unit of Measures can be found on the UOM_CLASS table. If the Threshold Limit Type is 'A' or 'P', then Threshold Limit Unit of Measure has to be blank. If the Threshold Limit Type is 'Q', Threshold Limit Unit of Measure must not be blank. If Threshold Limit Type is blank, Threshold Limit Unit of Measure must be blank.
	Threshold Value Type	Char(6)	Blank (space character string)	Identifies whether the discount values associated with the thresholds will be set up as qty values, currency amount values, percentages or fixed amounts. Valid values are 'Q' for qty, 'A' for currency amount, 'P' for percentage or 'F' for fixed amount. Qty threshold value (buy/get) deals are only allowed on off-invoice discounts. Deal threshold value types will be held on the codes table under a code type of 'DLL2'. It must be NULL for an M-type deal (vendor funded markdown). If Billing Type is 'BD' or 'BC', then the Threshold Value Type must not be 'Q'.
	Buy Item	Char(25)	Blank (space character string)	Identifies the item that must be purchased for a quantity threshold-type discount. This value is required for quantity threshold value type discounts. Otherwise it has to be blank.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Get Type	Char(6)	Blank (space character string)	Identifies the type of the 'get' discount for a quantity threshold-type (buy/get) discount. Valid values include 'X' (free), 'P' (percent), 'A' (amount) and 'F' (fixed amount). They are held on the codes table under a code type of 'DQGT'. This value is required for quantity threshold value deals. Otherwise it has to be blank.
	Get Value	Number(20,4)	All 0s.	Identifies the value of the 'get' discount for a quantity threshold-type (buy/get) discount that is not a 'free goods' deal. The Get Type above identifies the type of this value. This value is required for quantity threshold value type deals that are not a Get Type of free. Otherwise it has to be 0. If Get Type is ‘P’, ‘A’ or ‘F’, then Get Value must not be blank. If the Get Type is ‘X’ or blank, then Get Value has to be blank.
	Buy Item Quantity	Number(12,4)	All 0s.	Identifies the quantity of the threshold 'buy' item that must be ordered to qualify for the 'free' item. This value is required for quantity threshold value type discounts. Otherwise it has to be 0.
	Recursive Indicator	Char(1)	REQUIRED	For 'buy/get free' discounts, indicates if the quantity threshold discount is only for the first 'buy amt.' purchased (e.g. for the first 10 purchased, get 1 free), or if a free item will be given for every multiple of the 'buy amt' purchased on the order (e.g. for each 10 purchased, get 1 free). Valid values are 'Y' for yes or 'N' for no. If the Get Type is blank, then Recursive Indicator has to be ‘N’.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Buy Item Order Target Quantity	Number(12,4)	All 0s.	Indicates the targeted purchase level for all locations on a purchase order. This is the target level that will be used for future calculation of net cost. This value is required for quantity threshold value type deals. Otherwise it has to be 0.
	Average Buy Item Order Target Quantity Per Location	Number(12,4)	All 0s.	Indicates the average targeted purchase level per location on the deal. This value will be used in future cost calculations. This value is required for quantity threshold value type deals. Otherwise it has to be 0.
	Get Item	Char(25)	Blank (space character string)	Identifies the 'get' item for a quantity threshold-type (buy/get) discount. This value is required for quantity threshold value deals. Otherwise it has to be blank. If Get Type is ‘P’, ‘A’, ‘F’ or ‘X’, then Get Item must not be blank. If the Get Type is blank, then Get Item has to be blank.
	Get Quantity	Number(12,4)	All 0s.	Identifies the quantity of the identified 'get' item that will be given at the specified 'get' discount if the 'buy amt' of the buy item is purchased. This value is required for quantity threshold value type discounts. Otherwise it has to be 0. If Get Type is ‘P’, ‘A’, ‘F’ or ‘X’, then Get Quantity must not be 0. If the Get Type is blank, then Get Quantity has to be 0.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Free Item Unit Cost	Number(20,4)	All 0s.	For 'buy/get free' discounts, identifies the unit cost of the threshold 'free' item that will be used in calculating the prorated qty. discount. It will default to the item/supplier cost, but can be modified based on the agreement with the supplier. It must be greater than zero as this is the cost that would normally be charged for the goods if no deal applied. If Get Type is 'P', 'A', 'F' or blank, then Free Item Unit Cost must be 0. If the Get Type is 'X', then Free Item Unit Cost must not be 0.
	Transaction Level Discount Indicator	Char(1)	REQUIRED	Indicates if the discount is a transaction-level discount (e.g. 10% across an entire PO). Valid Values are 'Y' or 'N'. If set to 'Y', Deal Class has to be 'CU' and Billing Type has to be 'OI'. No DIDTL or PPDTL records may be present for a Transaction Level Discount DCDTL record.
	Rebate Indicator	Char(1)	REQUIRED	Indicates if the deal component is a rebate. Deal components can only be rebates for bill-back billing types. Valid values are 'Y' for yes or 'N' for no. If Billing Type is 'OI', then Rebate Indicator must be 'N'.
	Rebate Active Date	Char(14)	Blank (space character string)	If the rebate becomes active on a different date than the deal active date, this field will hold that date. If this field is NULL for a rebate line, it will be assumed that the rebate becomes active on the deal active date.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Rebate Calculation Type	Char(6)	Blank (space character string)	Indicates if the rebate should be calculated using linear or scalar calculation methods. Valid values are 'L' for linear or 'S' for scalar. This field will be required if the rebate indicator is 'Y'. Rebate calculation types will be held on the codes table under a code type of 'DLCT'. If Rebate Indicator is 'Y', then Rebate Calculation Type must not be blank. Otherwise it has to be blank.
	Growth Rebate Indicator	Char(1)	REQUIRED	Indicates if the rebate is a growth rebate, meaning it is calculated and applied based on an increase in purchases or sales over a specified period of time. Valid values are 'Y' for yes or 'N' for no. If Rebate Indicator is 'N', then Growth Rebate Indicator must be ‘N’.
	Historical Comparison Start Date	Char(14)	Blank (space character string)	The first date of the historical period against which growth will be measured in this growth rebate. Note performance and the rebate amount are not calculated - this field is for informational/reporting purposes only. If Growth Rebate Indicator is 'Y', then Historical Comparison Start Date must not be blank. Otherwise it must be blank.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Historical Comparison End Date	Char(14)	Blank (space character string)	The last date of the historical period against which growth will be measured in this growth rebate. Note performance and the rebate amount are not calculated - this field is for informational/reporting purposes only. If Growth Rebate Indicator is 'Y', then Historical Comparison End Date must not be blank. Otherwise it must be blank.
	Current Comparison Start Date	Char(14)	Blank (space character string)	The first date of the current/future period during which growth will be measured in this growth rebate. Note performance and the rebate amount are not calculated - this field is for informational/reporting purposes only. If Growth Rebate Indicator is 'Y', then Current Comparison Start Date must not be blank. Otherwise it must be blank.
	Current Comparison End Date	Char(14)	Blank (space character string)	The last date of the current/future period during which growth will be measured in this growth rebate. Note performance and the rebate amount are not calculated - this field is for informational/reporting purposes only. If Growth Rebate Indicator is 'Y', then Current Comparison End Date must not be blank. Otherwise it must be blank.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Rebate Purchases or Sales Application Indicator	Char(6)	Blank (space character string)	Indicates if the rebate should be applied to purchases or sales. Valid values are 'P' for purchases or 'S' for sales. It will be required if the rebate indicator is 'Y'. Rebate purchase/sales indicators will be held on the codes table under a code type of 'DLRP'. If the Rebate Indicator is 'Y', then the Rebate Purchases or Sales Application Indicator must not be blank. Otherwise it has to be blank.
	Comments	Char(2000)	Blank (space character string)	Free-form comments entered with the deal component.
Transaction Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail)
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal sub loop.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Detail Record Type	Char(5)	DIDTL	Identifies file record type of sub loop as Deal Component Item-location Detail.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload deal item-location details.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Merchandise Level	Char(6)	REQUIRED	Indicates what level of the merchandise hierarchy the record is at. Valid values include '1' for company-wide (all items), '2' for division, '3' for group, '4' for dept, '5' for class, '6' for subclass, '7' for line, '8' for line/diff 1, '9' for line/diff 2, '10 for line/diff 3, '11 for line/diff 4, and '12' for item. These level types will be held on the codes table under a code type of 'DIML'.
	Company Indicator	Char(1)	REQUIRED	Indicates if the deal component is applied company-wide (e.g. all items in the system will be included in the discount or rebate). Valid values are 'Y' for yes and 'N' for no.
	Division	Number(4)	Blank (space character string).	ID of the division included in or excluded from the deal component. Valid values are on the DIVISION table. If Group is not blank, then Division must not be blank. If Merchandise Level is 2, then Division must not be blank and Group, Department, Class and Subclass must be blank.
	Group	Number(4)	Blank (space character string).	ID of the group included in or excluded from the deal component. Valid values are on the GROUPS table. If Department is not blank, then Group must not be blank. If Merchandise Level is 3, then Group must not be blank and Department, Class and Subclass must be blank.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Department	Number(4)	Blank (space character string).	ID of the department included in or excluded from the deal component. Valid values are on the DEPS table. If Class is not blank, then Department must not be blank. If Merchandise Level is 4, then Department must not be blank and Class and Subclass must be blank.
	Class	Number(4)	Blank (space character string).	ID of the class included in or excluded from the deal component. Valid values are on the CLASS table. If Subclass is not blank, then Class must not be blank. If Merchandise Level is 5, then Class must not be blank and Subclass must be blank.
	Subclass	Number(4)	Blank (space character string).	ID of the subclass included in or excluded from the deal component. Valid values are on the SUBCLASS table. If Merchandise Level is 6 or more than 6, then Subclass must not be blank.
	Item Parent	Char(25)	Blank (space character string)	Alphanumeric value that uniquely identifies the item/group at the level above the item. This value must exist as an item in another row on the item_master table. If Merchandise Level is 7, then Item Parent or Item Grandparent must not be blank (at least one of them has to be given).

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Item Grandparent	Char(25)	Blank (space character string)	Alphanumeric value that uniquely identifies the item/group two levels above the item. This value must exist as both an item and an item parent in another row on the item_master table. If Merchandise Level is 7, then Item Parent or Item Grandparent must not be blank (at least one of them has to be given).
	Diff 1	Char(10)	Blank (space character string)	Diff_group or diff_id that differentiates the current item from its item_parent. If Item Grandparent and Item Parent are blank, then Diff 1 must be blank. If Merchandise Level is 8, then Diff 1 must not be blank.
	Diff 2	Char(10)	Blank (space character string)	Diff_group or diff_id that differentiates the current item from its item_parent. If Item Grandparent and Item Parent are blank, then Diff 2 must be blank. If Merchandise Level is 9, then Diff 2 must not be blank.
	Diff 3	Char(10)	Blank (space character string)	Diff_group or diff_id that differentiates the current item from its item_parent. If Item Grandparent and Item Parent are blank, then Diff 3 must be blank. If Merchandise Level is 10, then Diff 3 must not be blank.
	Diff 4	Char(10)	Blank (space character string)	Diff_group or diff_id that differentiates the current item from its item_parent. If Item Grandparent and Item Parent are blank, then Diff 4 must be blank. If Merchandise Level is 11, then Diff 4 must not be blank.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Organizational Level	Char(6)	Blank (space character string)	Indicates what level of the organizational hierarchy the record is at. Valid values include '1' for chain, '2' for area, '3' for region, '4' for district and '5' for location. These level types will be held on the codes table under a code type of 'DIOL'. If company indicator is N, this must not be blank. If location type is warehouse or location list, this must be 5.
	Chain	Number(4)	Blank (space character string).	ID of the chain included in or excluded from the deal component. Valid values are on the CHAIN table. If org. level is 1, this field must not be blank.
	Area	Number(4)	Blank (space character string).	ID of the area included in or excluded from the deal component. Valid values are on the AREA table. If org. level is 2, this field and chain must not be blank.
	Region	Number(4)	Blank (space character string).	ID of the region included in or excluded from the deal component. Valid values are on the REGION table. If org. level is 3, this field, area, and chain must not be blank.
	District	Number(4)	Blank (space character string).	ID of the district included in or excluded from the deal component. Valid values are on the DISTRICT table. If org. level is 4, then this field, region, area, and chain must not be blank.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Location	Number(10)	Blank (space character string).	ID of the location included in or excluded from the deal component. Valid values are on the STORE, WH, or LOC_LIST_HEAD table. If org. level is 5, this field must not be blank. Chain, area, region, and district should be blank if the loc_type is L or W. If the loc_type is S, then they all must not be blank. If Location Type is not blank, then Location must not be blank. Otherwise it has to be blank.
	Origin Country Identifier	Char(3)	Blank (space character string)	Origin country of the item that the deal component should apply to.
	Location Type	Char(1)	Blank (space character string)	Type of the location referenced in the location field. Valid values are 'S' and 'W'. Location types will be held on the codes table under the code type 'LOC3'. If location is blank then this field has to be blank also.
	Item	Char(25)	Blank (space character string)	Unique alphanumeric value that identifies the item. If Merchandise Level is 12, then Item must not be blank.
	Exclusion Indicator	Char(1)	REQUIRED	Indicates if the deal component item/location line is included in the deal component or excluded from it. Valid values are 'Y' for yes or 'N' for no.
	Reference Line	Number(10)	REQUIRED	This value determines which line in the input file this item-loc record belongs to. See the section XIII Design Assumptions for more explanation on how this field should be populated.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
Transaction Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail)
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal sub loop.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Detail Record Type	Char(5)	PPDTL	Identifies file record type of sub loop as Proof of Performance Detail.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload deal proof of performance details.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Deal Sub Item	Char(25)		Specific transaction level (or below) item that's proof of performance is being measured. This can be populated when the deal itself is on a case UPC but the proof of performance is on an individual selling unit.
	Proof of Performance Type	Char(6)	REQUIRED	Code that identifies the proof of performance type (i.e. term is that the item must be displayed on an end cap for 28 days - the pop_type is code 'E' for end cap display). Valid values for this field are stored in the code_type = 'PPT'. This field is required by the database.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Proof of Performance Value	Number(20,4)	All 0s.	Value that describes the term of the proof of performance type (i.e. term is that the item must be displayed on an end cap for 28 days - the pop_value is 28). This field is required by the database if the record has a pop_value_type. If Proof of Performance Value is not blank, then Proof of Performance Value Type must not be blank. If Proof of Performance Value is blank, then Proof of Performance Value Type must be blank.
	Proof of Performance Value Type	Char(6)	Blank (space character string)	Value that describes the type of the pop_value (i.e. term is that the item must be displayed on an end cap for 28 days - the pop_value_type is the code 'D' for days). Valid values for this field are stored in the code_type = 'PPVT'. This field is required by the database if the record has a pop_value. If Proof of Performance Value is not blank, then Proof of Performance Value Type must not be blank. If Proof of Performance Value is blank, then Proof of Performance Value Type must be blank.
	Vendor Recommended Start Date	Char(14)	Blank (space character string)	This column holds the date that the vendor recommends that the POP begin.
	Vendor Recommended End Date	Char(14)	Blank (space character string)	This column holds the date that the vendor recommends that the POP end.
	Planned Start Date	Char(14)	Blank (space character string)	This column holds the date that the merchandiser/category manager plans to begin the POP.
	Planned End Date	Char(14)	Blank (space character string)	This column holds the date that the merchandiser/category manager plans to end the POP.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Comment	Char(255)	Blank (space character string)	Free-form comments.
	Reference Line	Number(10)	REQUIRED	This value determines which line in the input file this Proof of Performance record belongs to. See the Assumptions section for more explanation on how this field should be populated.
Transacti on Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail)
Transacti on Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal sub loop.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Detail Record Type	Char(5)	DTDTL	Identifies file record type of sub loop as Deal Component Threshold Detail.
Transacti on Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload deal threshold details.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Lower Limit	Number(20,4)	REQUIRED	Lower limit of the deal component. This is the minimum value that must be met in order to get the specified discount. This value will be either a currency amount or quantity value, depending on the value in the deal_detail.threshold_limit_type field of this deal component (Threshold Value Type field of the DCDTL record that this DTDTL record belongs to as specified in the reference line field).
	Upper Limit	Number(20,4)	REQUIRED	Upper limit of the deal component. This is the maximum value for which the specified discount will apply. This value will be either a currency amount or quantity value, depending on the value in the deal_detail.threshold_limit_type field of this deal component (Threshold Value Type field of the DCDTL record that this DTDTL record belongs to as specified in the reference line field).
	Value	Number(20,4)	REQUIRED	Value of the discount that will be given for meeting the specified thresholds for this deal component. This value will be either a currency amount or quantity value, depending on the value in the deal_detail.threshold_value_type field of this deal component (Threshold Value Type field of the DCDTL record that this DTDTL record belongs to as specified in the reference line field).
	Target Level Indicator	Char(1)	REQUIRED	Indicates if a threshold level is the targeted purchase or sales level for a deal component. This indicator will be used for cost calculations. Valid values are 'Y' for yes and 'N' for no.
	Reference Line	Number(10)	REQUIRED	This value determines which line in the input file this Threshold record belongs to. See the Assumptions section for more explanation on how this field should be populated.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
Transaction Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail)
File Trailer	File Line Identifier	Char(5)	FTAIL	Identifies file record type (the end of the input file).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	File Record Counter	Numeric ID(10)	Sequential number Created by program.	Number of records/transactions in current file (only records between head & tail)

Output Specifications

‘Table-To-Table’

Reject File

Record Name	Field Name	Field Type	Default Value	Description/Constraints
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type (the beginning of the input file).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	File Type Definition	Char(5)	EDIDU	Identifies file as ‘EDI Deals Upload’
	File Create Date	Char(14)	Create date	current date, formatted to ‘YYYYMMDDHH24MISS’.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal header.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Detail Record Type	Char(5)	DHDTL	Identifies file record type Deal Header. This record MUST BE FOLLOWED BY ONE AND ONLY ONE REQUIRED TDETL RECORD that holds the deal head information.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload a new deal.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Partner Type	Char(6)	REQUIRED	Type of the partner the deal applies to. Valid values are ‘S’ for a supplier, 'S1' for supplier hierarchy level 1 (e.g. manufacturer), 'S2' for supplier hierarchy level 2 (e.g. distributor) and 'S3' for supplier hierarchy level 3 (e.g. wholesaler). Descriptions of these codes will be held on the codes table under a code_type of 'SUHL'. Information pertaining to a single deal has to belong to the same supplier, since a deal may have only one supplier hierarchy associated with it. Only items with the same supplier hierarchy can be on the same deal. Supplier hierarchy is stored at an item / supplier / country / location level.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Partner Id	Char(10)	Blank (space character string)	Level of supplier hierarchy (e.g. manufacturer, distributor or wholesaler), set up as a partner in the PARTNER table, used for assigning rebates by a level other than supplier. Rebates at this level will include all eligible supplier/item/country records assigned to this supplier hierarchy level. This field is required if the Partner Type field was set to 'S1', 'S2' or 'S3'. This field must be blank if the Partner Type field was set to 'S'.
	Supplier	Number(10)	Blank (space character string)	Deal supplier's number. This supplier can be at any level of supplier hierarchy. This field is required if the Partner Type field was set to 'S'. This field must be blank if the Partner Type field was set to 'S1', 'S2' or 'S3'.
	Type	Char(6)	REQUIRED	Type of the deal. Valid values are A for annual deal, P for promotional deal, O for PO-specific deal or M for vendor-funded markdown. Deal types will be held on the codes table under a code type of 'DLHT'.
	Currency Code	Char(3)	Blank (space character string)	Currency code of the deal's currency. All costs on the deal will be held in this currency. If Type is 'O', 'P' or 'A', then Currency Code may not be blank. Currency Code has to be blank if Type is 'M'.
	Active Date	Char(14)	REQUIRED	Date the deal will become active. This date will determine when deal components begin to be factored into item costs. For a PO-specific deal, the active_date will be the order's written date.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Close Date	Char(14)	Blank (space character string)	Date the deal will/did end. This date determines when deal components are no longer factored into item costs. It is optional for annual deals, required for promotional deals. It will be left NULL for PO-specific deals. Close Date must not be blank if Type is 'P' or 'M'. Close Date has to be blank if Type is 'O'.
	External Reference Number	Char(30)	Blank (space character string)	Any given external reference number that is associated with the deal.
	Order Number	Number(8)	Blank (space character string)	Order the deal applies to, if the deal is PO-specific.
	Recalculate Approved Orders	Char(1)	REQUIRED	Indicates if approved orders should be recalculated based on this deal once the deal is approved. Valid values are Y for yes or N for no. Valid values are 'Y' and 'N'.
	Comments	Char(2000)	Blank (space character string)	Free-form comments entered with the deal.
Transaction Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail). For DHDTL TDETL records this will always be 1!
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal sub loop.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Detail Record Type	Char(5)	DCDTL	Identifies file record type of sub loop as Deal Component Detail.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload deal components.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Deal Component Type	Char(6)	REQUIRED	Type of the deal component, user-defined and stored on the DEAL_COMP_TYPE table.
	Application Order	Number(10)	Blank (space character string)	Number indicating the order in which the deal component should be applied with respect to any other deal components applicable to the item within the deal. This number will be unique across all deal components within the deal. It must be NULL for an M-type deal (vendor funded markdown).
	Billing Type	Char(6)	REQUIRED	Billing type of the deal component. Valid values are 'OI' for off-invoice, 'BD' for bill-back with debit memo or 'BC' for bill-back with credit note request. Billing types will be held on the codes table under a code type of 'DLBT'.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Bill Back Period	Char(6)	Blank (space character string)	Code that identifies the bill-back period for the deal component. This field will only be populated for billing types of 'BD' or 'BC'. Bill back period codes will be user-defined and stored on the BILL_BACK_PERIOD table. If Billing Type is 'BD' or 'BC' then Bill Back Period must not be blank, otherwise it has to be blank.
	Collect Start Date	Char(14)	Blank (space character string)	Date that collection of the bill-back should begin. If Billing Type is 'BD' or 'BC' then Collect Start Date must not be blank, otherwise it has to be blank.
	Collect End Date	Char(14)	Blank (space character string)	Date that collection of the bill-back should end. If Billing Type is 'BD' or 'BC' then Collect End Date must not be blank, otherwise it has to be blank.
	Deal Application Timing	Char(6)	Blank (space character string)	Indicates when the deal component should be applied - at PO approval or time of receiving. Valid values are 'O' for PO approval, 'R' for receiving. These values will be held on the codes tables under a code type of 'AALC'. It must be NULL for an M-type deal (vendor funded markdown).
	Cost Application Level Indicator	Char(6)	Blank (space character string)	Indicates what cost bucket the deal component should affect. Valid values are 'N' for net cost, 'NN' for net net cost and 'DNN' for dead net net cost. These values will be held on the codes tables under a code type of 'DLCA'. It must be NULL for an M-type deal (vendor funded markdown).

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Pricing Cost Indicator	Char(1)	REQUIRED	Identifies deal components that should be included when calculating a pricing cost. Valid values are ‘Y’es and ‘N’o.
	Deal Class	Char(6)	Blank (space character string)	Identifies the calculation class of the deal component. Valid values are 'CU' for cumulative (discounts are added together and taken off as one lump sum), 'CS' for cascade (discounts are taken one at a time with subsequent discounts taken off the result of the previous discount) and 'EX' for exclusive (overrides all other discounts). 'EX' type deal components are only valid for promotional deals. Deal classes will be held on the codes table under a code type of 'DLCL'. It must be NULL for an M-type deal (vendor funded markdown).
	Threshold Limit Type	Char(6)	Blank (space character string)	Identifies whether thresholds will be set up as qty values, currency amount values or percentages (growth rebates only). Valid values are 'Q' for qty, 'A' for currency amount or 'P' for percentage. Threshold limit types will be held on the codes table under a code type of 'DLLT'. It must be NULL for an M-type deal (vendor funded markdown) or if the threshold value type is ‘Q’ (buy/get deals). If Growth Rebate Indicator is 'Y', then the Threshold Limit Type has to be 'P', otherwise 'Q', 'A' or NULL.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Threshold Limit Unit of Measure	Char(4)	Blank (space character string)	Unit of measure of the threshold limits, if the limit type is quantity. Only Unit of Measures with a UOM class of 'VOL' (volume), 'MASS' or 'QTY' (quantity) can be used in this field. Valid Unit of Measures can be found on the UOM_CLASS table. If the Threshold Limit Type is 'A' or 'P', then Threshold Limit Unit of Measure has to be blank. If the Threshold Limit Type is 'Q', Threshold Limit Unit of Measure must not be blank. If Threshold Limit Type is blank, Threshold Limit Unit of Measure must be blank.
	Threshold Value Type	Char(6)	Blank (space character string)	Identifies whether the discount values associated with the thresholds will be set up as qty values, currency amount values, percentages or fixed amounts. Valid values are 'Q' for qty, 'A' for currency amount, 'P' for percentage or 'F' for fixed amount. Qty threshold value (buy/get) deals are only allowed on off-invoice discounts. Deal threshold value types will be held on the codes table under a code type of 'DLL2'. It must be NULL for an M-type deal (vendor funded markdown). If Billing Type is 'BD' or 'BC', then the Threshold Value Type must not be 'Q'.
	Buy Item	Char(25)	Blank (space character string)	Identifies the item that must be purchased for a quantity threshold-type discount. This value is required for quantity threshold value type discounts. Otherwise it has to be blank.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Get Type	Char(6)	Blank (space character string)	Identifies the type of the 'get' discount for a quantity threshold-type (buy/get) discount. Valid values include 'X' (free), 'P' (percent), 'A' (amount) and 'F' (fixed amount). They are held on the codes table under a code type of 'DQGT'. This value is required for quantity threshold value deals. Otherwise it has to be blank.
	Get Value	Number(20,4)	All 0s.	Identifies the value of the 'get' discount for a quantity threshold-type (buy/get) discount that is not a 'free goods' deal. The Get Type above identifies the type of this value. This value is required for quantity threshold value type deals that are not a Get Type of free. Otherwise it has to be 0. If Get Type is 'P', 'A' or 'F', then Get Value must not be blank. If the Get Type is 'X' or blank, then Get Value has to be blank.
	Buy Item Quantity	Number(12,4)	All 0s.	Identifies the quantity of the threshold 'buy' item that must be ordered to qualify for the 'free' item. This value is required for quantity threshold value type discounts. Otherwise it has to be 0.
	Recursive Indicator	Char(1)	REQUIRED	For 'buy/get free' discounts, indicates if the quantity threshold discount is only for the first 'buy amt.' purchased (e.g. for the first 10 purchased, get 1 free), or if a free item will be given for every multiple of the 'buy amt' purchased on the order (e.g. for each 10 purchased, get 1 free). Valid values are 'Y' for yes or 'N' for no. If the Get Type is blank, then Recursive Indicator has to be 'N'.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Buy Item Order Target Quantity	Number(12,4)	All 0s.	Indicates the targeted purchase level for all locations on a purchase order. This is the target level that will be used for future calculation of net cost. This value is required for quantity threshold value type deals. Otherwise it has to be 0.
	Average Buy Item Order Target Quantity Per Location	Number(12,4)	All 0s.	Indicates the average targeted purchase level per location on the deal. This value will be used in future cost calculations. This value is required for quantity threshold value type deals. Otherwise it has to be 0.
	Get Item	Char(25)	Blank (space character string)	Identifies the 'get' item for a quantity threshold-type (buy/get) discount. This value is required for quantity threshold value deals. Otherwise it has to be blank. If Get Type is ‘P’, ‘A’, ‘F’ or ‘X’, then Get Item must not be blank. If the Get Type is blank, then Get Item has to be blank.
	Get Quantity	Number(12,4)	All 0s.	Identifies the quantity of the identified 'get' item that will be given at the specified 'get' discount if the 'buy amt' of the buy item is purchased. This value is required for quantity threshold value type discounts. Otherwise it has to be 0. If Get Type is ‘P’, ‘A’, ‘F’ or ‘X’, then Get Quantity must not be 0. If the Get Type is blank, then Get Quantity has to be 0.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Free Item Unit Cost	Number(20,4)	All 0s.	For 'buy/get free' discounts, identifies the unit cost of the threshold 'free' item that will be used in calculating the prorated qty. discount. It will default to the item/supplier cost, but can be modified based on the agreement with the supplier. It must be greater than zero as this is the cost that would normally be charged for the goods if no deal applied. If Get Type is 'P', 'A', 'F' or blank, then Free Item Unit Cost must be 0. If the Get Type is 'X', then Free Item Unit Cost must not be 0.
	Transaction Level Discount Indicator	Char(1)	REQUIRED	Indicates if the discount is a transaction-level discount (e.g. 10% across an entire PO). Valid Values are 'Y' or 'N'. If set to 'Y', Deal Class has to be 'CU' and Billing Type has to be 'OI'. No DIDTL or PPDTL records may be present for a Transaction Level Discount DCDTL record.
	Rebate Indicator	Char(1)	REQUIRED	Indicates if the deal component is a rebate. Deal components can only be rebates for bill-back billing types. Valid values are 'Y' for yes or 'N' for no. If Billing Type is 'OI', then Rebate Indicator must be 'N'.
	Rebate Active Date	Char(14)	Blank (space character string)	If the rebate becomes active on a different date than the deal active date, this field will hold that date. If this field is NULL for a rebate line, it will be assumed that the rebate becomes active on the deal active date.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Rebate Calculation Type	Char(6)	Blank (space character string)	Indicates if the rebate should be calculated using linear or scalar calculation methods. Valid values are 'L' for linear or 'S' for scalar. This field will be required if the rebate indicator is 'Y'. Rebate calculation types will be held on the codes table under a code type of 'DLCT'. If Rebate Indicator is 'Y', then Rebate Calculation Type must not be blank. Otherwise it has to be blank.
	Growth Rebate Indicator	Char(1)	REQUIRED	Indicates if the rebate is a growth rebate, meaning it is calculated and applied based on an increase in purchases or sales over a specified period of time. Valid values are 'Y' for yes or 'N' for no. If Rebate Indicator is 'N', then Growth Rebate Indicator must be ‘N’.
	Historical Comparison Start Date	Char(14)	Blank (space character string)	The first date of the historical period against which growth will be measured in this growth rebate. Note performance and the rebate amount are not calculated - this field is for informational/reporting purposes only. If Growth Rebate Indicator is 'Y', then Historical Comparison Start Date must not be blank. Otherwise it must be blank.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Historical Comparison End Date	Char(14)	Blank (space character string)	The last date of the historical period against which growth will be measured in this growth rebate. Note performance and the rebate amount are not calculated - this field is for informational/reporting purposes only. If Growth Rebate Indicator is 'Y', then Historical Comparison End Date must not be blank. Otherwise it must be blank.
	Current Comparison Start Date	Char(14)	Blank (space character string)	The first date of the current/future period during which growth will be measured in this growth rebate. Note performance and the rebate amount are not calculated - this field is for informational/reporting purposes only. If Growth Rebate Indicator is 'Y', then Current Comparison Start Date must not be blank. Otherwise it must be blank.
	Current Comparison End Date	Char(14)	Blank (space character string)	The last date of the current/future period during which growth will be measured in this growth rebate. Note performance and the rebate amount are not calculated - this field is for informational/reporting purposes only. If Growth Rebate Indicator is 'Y', then Current Comparison End Date must not be blank. Otherwise it must be blank.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Rebate Purchases or Sales Application Indicator	Char(6)	Blank (space character string)	Indicates if the rebate should be applied to purchases or sales. Valid values are 'P' for purchases or 'S' for sales. It will be required if the rebate indicator is 'Y'. Rebate purchase/sales indicators will be held on the codes table under a code type of 'DLRP'. If the Rebate Indicator is 'Y', then the Rebate Purchases or Sales Application Indicator must not be blank. Otherwise it has to be blank.
	Comments	Char(2000)	Blank (space character string)	Free-form comments entered with the deal component.
Transacti on Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail)
Transacti on Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal sub loop.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Detail Record Type	Char(5)	DIDTL	Identifies file record type of sub loop as Deal Component Item-location Detail.
Transacti on Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload deal item-location details.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Merchandise Level	Char(6)	REQUIRED	Indicates what level of the merchandise hierarchy the record is at. Valid values include '1' for company-wide (all items), '2' for division, '3' for group, '4' for dept, '5' for class, '6' for subclass, '7' for line, '8' for line/diff 1, '9' for line/diff 2, '10' for line/diff 3, '11' for line/diff 4 and '12' for item. These level types will be held on the codes table under a code type of 'DIML'.
	Company Indicator	Char(1)	REQUIRED	Indicates if the deal component is applied company-wide (e.g. all items in the system will be included in the discount or rebate). Valid values are 'Y' for yes and 'N' for no.
	Division	Number(4)	Blank (space character string).	ID of the division included in or excluded from the deal component. Valid values are on the DIVISION table. If Group is not blank, then Division must not be blank. If Merchandise Level is 2, then Division must not be blank and Group, Department, Class and Subclass must be blank.
	Group	Number(4)	Blank (space character string).	ID of the group included in or excluded from the deal component. Valid values are on the GROUPS table. If Department is not blank, then Group must not be blank. If Merchandise Level is 3, then Group must not be blank and Department, Class and Subclass must be blank.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Department	Number(4)	Blank (space character string).	ID of the department included in or excluded from the deal component. Valid values are on the DEPS table. If Class is not blank, then Department must not be blank. If Merchandise Level is 4, then Department must not be blank and Class and Subclass must be blank.
	Class	Number(4)	Blank (space character string).	ID of the class included in or excluded from the deal component. Valid values are on the CLASS table. If Subclass is not blank, then Class must not be blank. If Merchandise Level is 5, then Class must not be blank and Subclass must be blank.
	Subclass	Number(4)	Blank (space character string).	ID of the subclass included in or excluded from the deal component. Valid values are on the SUBCLASS table. If Merchandise Level is 6 or more than 6, then Subclass must not be blank.
	Item Parent	Char(25)	Blank (space character string)	Alphanumeric value that uniquely identifies the item/group at the level above the item. This value must exist as an item in another row on the item_master table. If Merchandise Level is 7, then Item Parent or Item Grandparent must not be blank (at least one of them has to be given).

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Item Grandparent	Char(25)	Blank (space character string)	Alphanumeric value that uniquely identifies the item/group two levels above the item. This value must exist as both an item and an item parent in another row on the item_master table. If Merchandise Level is 7, then Item Parent or Item Grandparent must not be blank (at least one of them has to be given).
	Diff 1	Char(10)	Blank (space character string)	Diff_group or diff_id that differentiates the current item from its item_parent. If Item Grandparent and Item Parent are blank, then Diff 1 must be blank. If Merchandise Level is 8, then Diff 1 must not be blank.
	Diff 2	Char(10)	Blank (space character string)	Diff_group or diff_id that differentiates the current item from its item_parent. If Item Grandparent and Item Parent are blank, then Diff 2 must be blank. If Merchandise Level is 9, then Diff 2 must not be blank.
	Diff 3	Char(10)	Blank (space character string)	Diff_group or diff_id that differentiates the current item from its item_parent. If Item Grandparent and Item Parent are blank, then Diff 3 must be blank. If Merchandise Level is 10, then Diff 3 must not be blank.
	Diff 4	Char(10)	Blank (space character string)	Diff_group or diff_id that differentiates the current item from its item_parent. If Item Grandparent and Item Parent are blank, then Diff 4 must be blank. If Merchandise Level is 11, then Diff 4 must not be blank.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Organizational Level	Char(6)	Blank (space character string)	Indicates what level of the organizational hierarchy the record is at. Valid values include '1' for chain, '2' for area, '3' for region, '4' for district and '5' for location. These level types will be held on the codes table under a code type of 'DIOL'. If company indicator is N, this must not be blank. If location type is warehouse or location list, this must be 5.
	Chain	Number(4)	Blank (space character string).	ID of the chain included in or excluded from the deal component. Valid values are on the CHAIN table. If org. level is 1, this field must not be blank.
	Area	Number(4)	Blank (space character string).	ID of the area included in or excluded from the deal component. Valid values are on the AREA table. If org. level is 2, this field and chain must not be blank.
	Region	Number(4)	Blank (space character string).	ID of the region included in or excluded from the deal component. Valid values are on the REGION table. If org. level is 3, this field, area, and chain must not be blank.
	District	Number(4)	Blank (space character string).	ID of the district included in or excluded from the deal component. Valid values are on the DISTRICT table. If org. level is 4, then this field, region, area, and chain must not be blank.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Location	Number(10)	Blank (space character string).	ID of the location included in or excluded from the deal component. Valid values are on the STORE, WH, or LOC_LIST_HEAD table. If org. level is 5, this field must not be blank. Chain, area, region, and district should be blank if the loc_type is L or W. If the loc_type is S, then they all must not be blank. If Location Type is not blank, then Location must not be blank. Otherwise it has to be blank.
	Origin Country Identifier	Char(3)	Blank (space character string)	Origin country of the item that the deal component should apply to.
	Location Type	Char(1)	Blank (space character string)	Type of the location referenced in the location field. Valid values are 'S' and 'W'. Location types will be held on the codes table under the code type 'LOC3'. If location is blank then this field has to be blank also.
	Item	Char(25)	Blank (space character string)	Unique alphanumeric value that identifies the item. If Merchandise Level is 12, then Item must not be blank.
	Exclusion Indicator	Char(1)	REQUIRED	Indicates if the deal component item/location line is included in the deal component or excluded from it. Valid values are 'Y' for yes or 'N' for no.
	Reference Line	Number(10)	REQUIRED	This value determines which line in the input file this item-loc record belongs to. See the section XIII Design Assumptions for more explanation on how this field should be populated.
Transaction Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail)
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal sub loop.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Detail Record Type	Char(5)	PPDTL	Identifies file record type of sub loop as Proof of Performance Detail.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload deal proof of performance details.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Deal Sub Item	Char(25)		Specific transaction level (or below) item that's proof of performance is being measured. This can be populated when the deal itself is on a case UPC but the proof of performance is on an individual selling unit.
	Proof of Performance Type	Char(6)	REQUIRED	Code that identifies the proof of performance type (i.e. term is that the item must be displayed on an end cap for 28 days - the pop_type is code 'E' for end cap display). Valid values for this field are stored in the code_type = 'PPT'. This field is required by the database.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Proof of Performance Value	Number(20,4)	All 0s.	Value that describes the term of the proof of performance type (i.e. term is that the item must be displayed on an end cap for 28 days - the pop_value is 28). This field is required by the database if the record has a pop_value_type. If Proof of Performance Value is not blank, then Proof of Performance Value Type must not be blank. If Proof of Performance Value is blank, then Proof of Performance Value Type must be blank.
	Proof of Performance Value Type	Char(6)	Blank (space character string)	Value that describes the type of the pop_value (i.e. term is that the item must be displayed on an end cap for 28 days - the pop_value_type is the code 'D' for days). Valid values for this field are stored in the code_type = 'PPVT'. This field is required by the database if the record has a pop_value. If Proof of Performance Value is not blank, then Proof of Performance Value Type must not be blank. If Proof of Performance Value is blank, then Proof of Performance Value Type must be blank.
	Vendor Recommended Start Date	Char(14)	Blank (space character string)	This column holds the date that the vendor recommends that the POP begin.
	Vendor Recommended End Date	Char(14)	Blank (space character string)	This column holds the date that the vendor recommends that the POP end.
	Planned Start Date	Char(14)	Blank (space character string)	This column holds the date that the merchandiser/category manager plans to begin the POP.
	Planned End Date	Char(14)	Blank (space character string)	This column holds the date that the merchandiser/category manager plans to end the POP.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Comment	Char(255)	Blank (space character string)	Free-form comments.
	Reference Line	Number(10)	REQUIRED	This value determines which line in the input file this Proof of Performance record belongs to. See the Assumptions section for more explanation on how this field should be populated.
Transacti on Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail)
Transacti on Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type to upload a new deal sub loop.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Detail Record Type	Char(5)	DTDTL	Identifies file record type of sub loop as Deal Component Threshold Detail.
Transacti on Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type to upload deal threshold details.
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.

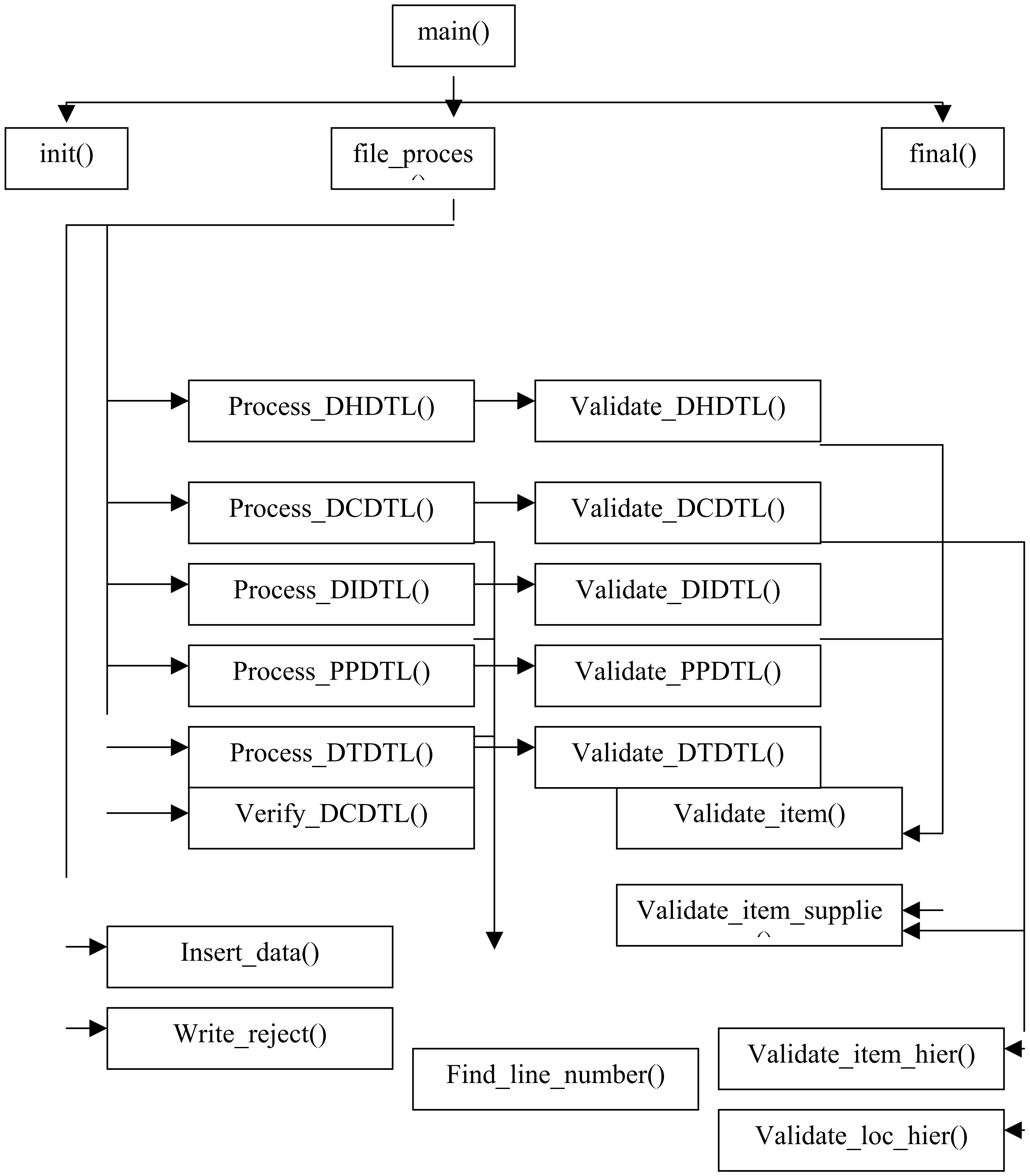
Record Name	Field Name	Field Type	Default Value	Description/Constraints
	Lower Limit	Number(20,4)	REQUIRED	Lower limit of the deal component. This is the minimum value that must be met in order to get the specified discount. This value will be either a currency amount or quantity value, depending on the value in the deal_detail.threshold_limit_type field of this deal component (Threshold Value Type field of the DCDTL record that this DTDTL record belongs to as specified in the reference line field).
	Upper Limit	Number(20,4)	REQUIRED	Upper limit of the deal component. This is the maximum value for which the specified discount will apply. This value will be either a currency amount or quantity value, depending on the value in the deal_detail.threshold_limit_type field of this deal component (Threshold Value Type field of the DCDTL record that this DTDTL record belongs to as specified in the reference line field).
	Value	Number(20,4)	REQUIRED	Value of the discount that will be given for meeting the specified thresholds for this deal component. This value will be either a currency amount or quantity value, depending on the value in the deal_detail.threshold_value_type field of this deal component (Threshold Value Type field of the DCDTL record that this DTDTL record belongs to as specified in the reference line field).
	Target Level Indicator	Char(1)	REQUIRED	Indicates if a threshold level is the targeted purchase or sales level for a deal component. This indicator will be used for cost calculations. Valid values are 'Y' for yes and 'N' for no.
	Reference Line	Number(10)	REQUIRED	This value determines which line in the input file this Threshold record belongs to. See the Assumptions section for more explanation on how this field should be populated.

Record Name	Field Name	Field Type	Default Value	Description/Constraints
Transaction Trailer	File Line Identifier	Char(5)	TTAIL	Identifies file record type (the end of the transaction detail).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	Transaction Record Counter	Numeric ID(6)	Sequential number Created by program.	Number of records/transactions in current transaction set (only records between thead & ttail)
File Trailer	File Line Identifier	Char(5)	FTAIL	Identifies file record type (the end of the input file).
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being read from input file.
	File Record Counter	Numeric ID(10)	Sequential number Created by program.	Number of records/transactions in current file (only records between head & tail)

Information from the input file is uploaded into the following Deals Management tables:

- DEAL_HEAD
- DEAL_DETAIL
- DEAL_THRESHOLD
- DEAL_ITEMLOC
- POP_TERMS_DEF

Function Level Description



Declare reject file line number counter(s) and structs to hold data fetched from input file until one deal is complete and to process validation. Declare global variables as needed. The following single record structs are needed:

- A struct to hold the FHEAD record.
- A struct to hold the THEAD record.
- A struct to hold the DHDTL record + system fields for the insert into RMS (eg: deal_id, line-number).

Declare sizeable structs to hold the potentially multi-line sub loops of a DHDTL record from the input file:

- A struct to hold TDETL of DCDTL records + system fields for the insert into RMS (eg: deal_id, line-number).
- A struct to hold TDETL of DIDTL records + system fields for the insert into RMS (eg: deal_id, line-number).
- A struct to hold TDETL of DTDTL records + system fields for the insert into RMS (eg: deal_id).
- A struct to hold TDETL of PPDTL records + system fields for the insert into RMS (eg: deal_id).

Main(): Standard RETEK main function.

- Log on to DATABASE.
- Calls init(), process() and final().
- Log appropriate messages for batch run based on return from above calls.

Init(): Handles restart/recovery initialization, populates global system variables for batch run. Opens input file, and reject output file.

Size_structs(): This function will size the four sizeable structs:

- A struct to hold TDETL of DCDTL records.
- A struct to hold TDETL of DIDTL records.
- A struct to hold TDETL of DTDTL records.
- A struct to hold TDETL of PPDTLrecords.

Resize_DCDTL_array(): grows the struct to hold TDETL of DCDTL records.

Resize_DIDTL_array(): grows the struct to hold TDETL of DDTL records.

Resize_DTDTL_array(): grows the struct to hold TDETL of DTDTL records.

Resize_PPDTL_array(): grows the struct to hold TDETL of PPDTL records.

File_process(): This function will call the rest of the functions necessary to process the input file while it loops through the input file.

- Call size_structs().
- Loop-get records from input file:

  - Save current file position of this record into a local variable in case something in the deal gets rejected and we need to write the entire deal and its sub-records to the reject file.
  - When FTAIL record is reached, break out of loop.
  - Get THEAD record, make sure it signals a DHDTL record to follow.
  - Call process_DHDTL().
  - Get THEAD record, make sure it signals a DCDTL record to follow.
  - Call process_DCDTL().
  - Get THEAD record, make sure it signals a DIDTL record to follow.
  - Call process_DIDTL().
  - Get THEAD record, make sure it signals a PPDTL record to follow.
  - Call process_PPDTL().
  - Get THEAD record, make sure it signals a DTDTL record to follow.
  - Call process_DTDTL().
  - If sub function returned code to signal a failed validation, set a local variable to reflect a non-fatal error.
  - Call insert_data() or write_reject() depending on whether any non-fatal errors have been recorded.
  - Set restart variables and force a commit.
- Return.

Process_DHDTL(): This function manages the processing of a DHDTL record.

- Fetch new deal_id.
- Get next line from input file, verify that it holds a TDETL.
- Insert DHDTL record from generic file-read buffer to the DHDTL struct, perform any nullpad or zeropad as necessary. Also insert line number of record into struct.
- Validate the DHDTL record by calling validate_DHDTL().
- If validation failed, write a non-fatal error and set return code so that file_process() knows it needs to dump records into the reject file.
- If validation of the DHDTL record passed, return code representing this status.

Validate_DHDTL(): Check information fetched from file to make sure it is complete and accurate.

- Make sure all required fields have a value other than the default.
- Make sure number fields are all numeric. (For details on what exactly has to appear and how, refer to the input file specification above. For field formatting, refer to the Assumptions section.)
- If an order number is given, verify it exists in RMS.

- If a supplier is given, verify it exists in RMS.

Process_DCDTL(): This function manages the processing of DCDTL records.

- Create deal_detail_id.
- Loop process:
    - Get next line from input file, verify that it holds a TDETL. If it is a TTAIL, break.
    - Insert DCDTL record from generic file-read buffer to the DCDTL struct, perform any nullpad or zeropad as necessary. Also insert line number of record, deal_id and deal_detail_id into struct.
    - Validate DCDTL record by calling validate_DCDTL().
    - If validation failed, write a non-fatal error and set return code so that file_process() knows it needs to dump records into the reject file.
    - If validation passed, go on processing the next record after incrementing counters and variables as necessary.

Validate_DCDTL(): Check information fetched from file to make sure it is complete and accurate.

- Make sure all required fields have a value other than the default.
- Make sure number fields are all numeric. (For details on what exactly has to appear and how, refer to the input file specification above. For field formatting, refer to the Assumptions section.)

Process_DIDTL(): This function manages the processing of DIDTL records.

- Create seq_no.
- Loop process:
    - Get next line from input file, verify that it holds a TDETL. If it is a TTAIL, break.
    - Insert DIDTL record from generic file-read buffer to the DIDTL struct, perform any nullpad or zeropad as necessary. Also insert line number of record and seq_no into struct.
    - Validate DIDTL record by calling validate_DIDTL().
    - Call find_line_number() to get the index of the record and which struct the record is in that this DIDTL record belongs to. At this level, verify that the line number was found in the DCDTL struct and copy over the deal_id and deal_detail_id into this struct. If the referenced DCDTL record that this DIDTL record belongs to has a type ‘M’ (vendor-funded markdown), there must be no DIDTL records. If the DCDTL record this DIDTL record is associated with is a Transaction Level Discount, write an error message.
    - If validation failed, write a non-fatal error and set return code so that file_process() knows it needs to dump records into the reject file.
    - If validation passed, go on processing the next record after incrementing counters and variables as necessary.

Validate_DIDTL(): Check information fetched from file to make sure it is complete and accurate.

- Make sure all required fields have a value other than the default.
- Make sure number fields are all numeric. (For details on what exactly has to appear and how, refer to the input file specification above. For field formatting, refer to the Assumptions section.)

Process_PPDTL(): This function manages the processing of PPDTL records.

- Create pop_def_seq_no.
- Loop process:
  - Get next line from input file, verify that it holds a TDETL. If it is a TTAIL, break.
  - Insert PPDTL record from generic file-read buffer to the PPDTL struct, perform any nullpad or zeropad as necessary. Also insert pop_def_seq_no into struct.
  - Validate PPDTL record by calling validate_PPDTL().
  - Call find_line_number() to get the index of the record and which struct the record is in that this PPDTL record belongs to. Copy over the deal_id, and deal_detail_id, and seq_no if available into this struct. If the referenced DCDTL record that this PPDTL record belongs to is a Transaction Level Deal, there must be no PPDTL records.
  - Validate deal_sub_item, which needs to be blank unless referenced record for this PPDTL record is a DIDTL record, in which case deal_sub_item must be a child or component item of the item in the referenced DIDTL record.
  - If validation failed, write a non-fatal error and set return code so that file_process() knows it needs to dump records into the reject file.
  - If validation passed, go on processing the next record after incrementing counters and variables as necessary.

Validate_PPDTL(): Check information fetched from file to make sure it is complete and accurate.

- Make sure all required fields have a value other than the default.
- Make sure number fields are all numeric. (For details on what exactly has to appear and how, refer to the input file specification above. For field formatting, refer to the Assumptions section.)
- Make sure that if both Vendor Recommended Start Date and Vendor Recommended End Date are given, then the start date is before the end date.
- Make sure that if both Planned Start Date and Planned End Date are given, then the start date is before the end date.
- Make sure that the current deal is not of type M (Vendor Funded Markdown). Such deals should have no PPDTL records.

Process_DTDTL(): This function manages the processing of DTDTL records.

- Loop process:
  - Get next line from input file, verify that it holds a TDETL. If it is a TTAIL, break.
  - Insert DTDTL record from generic file-read buffer to the DTDTL struct, perform any nullpad or zeropad as necessary.
  - Validate DTDTL record by calling validate_DTDTL().
  - Call find_line_number() to get the index of the record and which struct the record is in that this DTDTL record belongs to. At this level, verify that the line number was found in the DCDTL struct and copy over the deal_id, and deal_detail_id into this struct. If the referenced DCDTL record that this DTDTL record belongs to has a type 'M' (vendor-funded markdown), there must be no DTDTL records.
  - If validation failed, write a non-fatal error and set return code so that file_process() knows it needs to dump records into the reject file.
  - If validation passed, go on processing the next record after incrementing counters and variables as necessary.

Validate_DTDTL(): Check information fetched from file to make sure it is complete and accurate.

- Make sure all required fields have a value other than the default.
- Make sure number fields are all numeric. (For details on what exactly has to appear and how, refer to the input file specification above. For field formatting, refer to the Assumptions section.)

Validate_item(): Check if item is a valid item in RMS. Item has to be in approved status, may not be a buyer pack and has to be at or above transaction level.

- Fetch from item_master in RMS to verify the item identifier to be validated exists in RMS.

Validate_item_hier(): Check if item and its merchandise hierarchy is valid in RMS. Item has to be in approved status, may not be a buyer pack and has to be at or above transaction level.

- Fetch from item_master, deps, and groups in RMS to verify the item identifier and its merchandise hierarchy to be validated exists in RMS.

Validate_item_supplier(): Check if item and supplier relationship is valid in RMS.

Fetch from item_supplier in RMS to verify the item and supplier relationship is valid in RMS.

Validate_loc_hier(): Check if location and its organizational hierarchy is valid in RMS. Location may be a store or a physical warehouse only.

- Fetch a union from wh, store, district, region, and area in RMS to verify the location identifier and its organizational hierarchy to be validated exists in RMS.

Verify_DCDTL(): Check every DCDTL record and verify that:

- If DCDTL record has a billing type of OI (off-invoice) then there are at least one or more DIDTL records for this DCDTL record.
- If DCDTL record is a rebate then there are at least one or more DIDTL records for this DCDTL record.

Find_line_number(): Look in DHDTL then loop through the DCDTL and DIDTL arrays and look for the sequence number given in the argument of this function. Return the index and which array the line number was found in, or return an error message if the argument line number was not found in any of the arrays.

Insert_data(): Does a SQL looped insert from all the data structs with the loop count equaling the struct's record count. Inserts are performed in the following order: DHDTL, DCDTL, DIDTL, PPDTL, DTDTL.

Write_reject(): Writes to the reject file one deal record set with all its sub loop records from the input file.

Scheduling Considerations

This program should run as the first batch of the Deals batch cycle.

Locking Strategy

N/A.

Restart/Recovery

File based.

Performance Considerations

N/A.

Security Considerations

N/A.

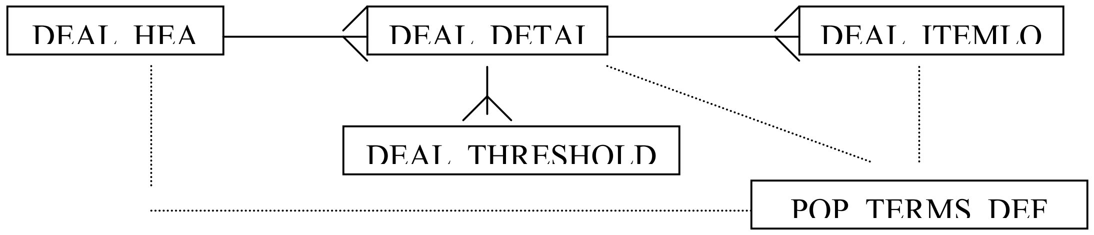
Design Assumptions

All fields that are of type Char in the input file format description above should be left justified and padded with the space character. If the field is not required and no value is being uploaded, the entire field should be padded with the space character.

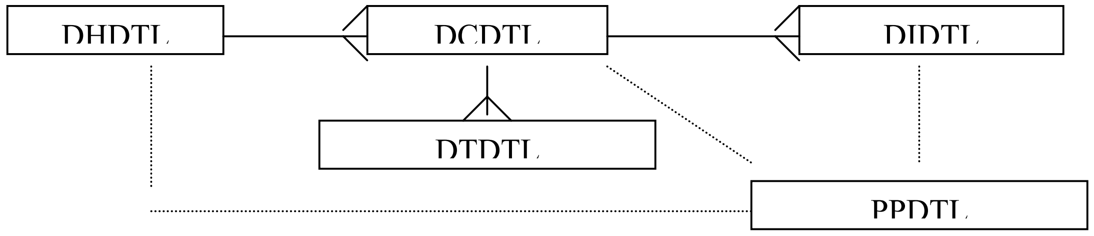
Date fields are always formatted 'YYYYMMDDHH24MISS' (Char(14)).

All fields that are of type Number in the input file format description above should be right justified and padded with 0s. If the field is not required and no value is being uploaded, the entire field should be padded with 0s.

Regarding the ordering of records in the input file, we require records to be looped in the same hierarchy as their corresponding RMS tables are in relation to each other:



Or in terms of the input file:



Note: Pop terms def may be associated with a single deal (FDETL), a deal component (TDETL of DCDTL) or a deal-item-location (TDETL of DIDTL) record. This ‘or’ type relationship is symbolized by a dashed ‘one-to-one’ line above.

Note: Buy/Get deals must have only the Buy and Get item(s) on the DEAL_ITEMLOC table, not more, nor less records.

The input file must have the following structure:

```
FHEAD
{
      THEAD of DHDTL    REQUIRED    for deal head record
            TDETL       REQUIRED    1 deal head record
      TTAIL             REQUIRED    end of deal head record
      THEAD of DCDTL    REQUIRED    for deal component records
      [
            TDETL       OPTIONAL    for deal component records
      ]
      TTAIL             REQUIRED    end of deal component records
      THEAD of DIDTL    REQUIRED    for item-loc records
      [
            TDETL       OPTIONAL    for item-loc records
      ]
      TTAIL             REQUIRED    end of item-loc records
      THEAD of PPDTL    REQUIRED    for proof of performance records
      [
            TDETL       OPTIONAL    for proof of performance records
      ]
      TTAIL             REQUIRED    end of proof of performance records
      THEAD of DTDTL    REQUIRED    for threshold records
      [
            TDETL       OPTIONAL    for threshold records
      ]
      TTAIL             REQUIRED    end of threshold records
}
FTAIL
```

The set between the curly brackets may be looped to upload multiple deals from the same file. Within each set, the TDETL records in angle brackets may be sub-looped as a sub-set of the main set.

One set equals one deal. For each set, the first TDETL record is used to upload the DEAL_HEAD (RMS table) table record into RMS. This record is required and maximum one may exist in each set, and it has to be the first THEAD-TDETL-TTAIL record group in the set.

The next THEAD-TDETL-TTAIL group holds the DEAL_DETAIL (RMS table for storing deal component records) records of the deal. Multiple TDETL records may be used between the THEAD and TTAIL records to upload multiple DEAL_DETAIL records for the same deal.

The next THEAD-TDETL-TTAIL group holds the DEAL_ITEMLOC (RMS table) records of the deal. Multiple TDETL records may be used between the THEAD and TTAIL records to upload multiple DEAL_ITEMLOC records for the same deal. Note that the line number field in these TDETL records refers to the input file line number of the DEAL_DETAIL record (DCDTL type, as signaled by preceding THEAD record,) to which this DEAL_ITEMLOC record belongs. This referenced line must be before current line and after current deal's DHDTL record (current deal is most recent DHDTL record in input file).

The next THEAD-TDETL-TTAIL group holds the POP_TERMS_DEF (RMS table for storing proof of performance records) records of the deal. Multiple TDETL records may be used between the THEAD and TTAIL records to upload multiple POP_TERMS_DEF records for the same deal. Note that the line number field in these TDETL records refers to the input file line number of the DEAL_HEAD, DEAL_DETAIL, or DEAL_ITEMLOC record (DHDTL, DCDTL, or DIDTL type, as signaled by preceding THEAD record,) to which this POP_TERMS_DEF record belongs. Note that a proof of performance record may belong to a DEAL_HEAD, DEAL_DETAIL, or DEAL_ITEMLOC record, not just the DEAL_DETAIL record. This referenced line must be before current line and at or after current deal's DHDTL record (current deal is most recent DHDTL record in input file).

The next THEAD-TDETL-TTAIL group holds the DEAL_THRESHOLD (RMS table) records of the deal. Multiple TDETL records may be used between the THEAD and TTAIL records to upload multiple DEAL_THRESHOLD records for the same deal. Note that the line number field in these TDETL records refers to the input file line number of the DEAL_DETAIL record (DCDTL type, as signaled by preceding THEAD record,) to which this DEAL_THRESHOLD record belongs. This referenced line must be before current line and after current deal's DHDTL record (current deal is most recent DHDTL record in input file).

Note that if a THEAD-TDETL-TTAIL group has no TDETL records, the THEAD-TTAIL records are still required in the input file, simply no TDETL records will appear between them. This is for explicitly signaling the fact that no such sub-records exist for the set.

Note: for an M type deal, no TDETL records of DIDTL, DTDTL or PPDTL type are allowed. (Vendor-funded markdowns have no records on the DEAL_ITEMLOC, DEAL_THRESHOLD or POP_TERMS_DEF RMS tables.)

Deal item insert [ditinsrt]

Functional Area

Complex Deals Management

Module Affected

Ditinsrt.pc – Deal Item Insert

Design Overview

This new batch program will populate the DEAL_SKU_TEMP table with all items that are on non vendor-funded, non PO-specific deals listed on the DEAL_QUEUE table and all items that fall within a hierarchy from these deals. It will get values for the entire merchandise and organizational hierarchies to populate DEAL_SKU_TEMP. The DEAL_SKU_TEMP table will then be used by precostcalc.pc and costcalc.pc to (re)calculate future costs for all listed items.

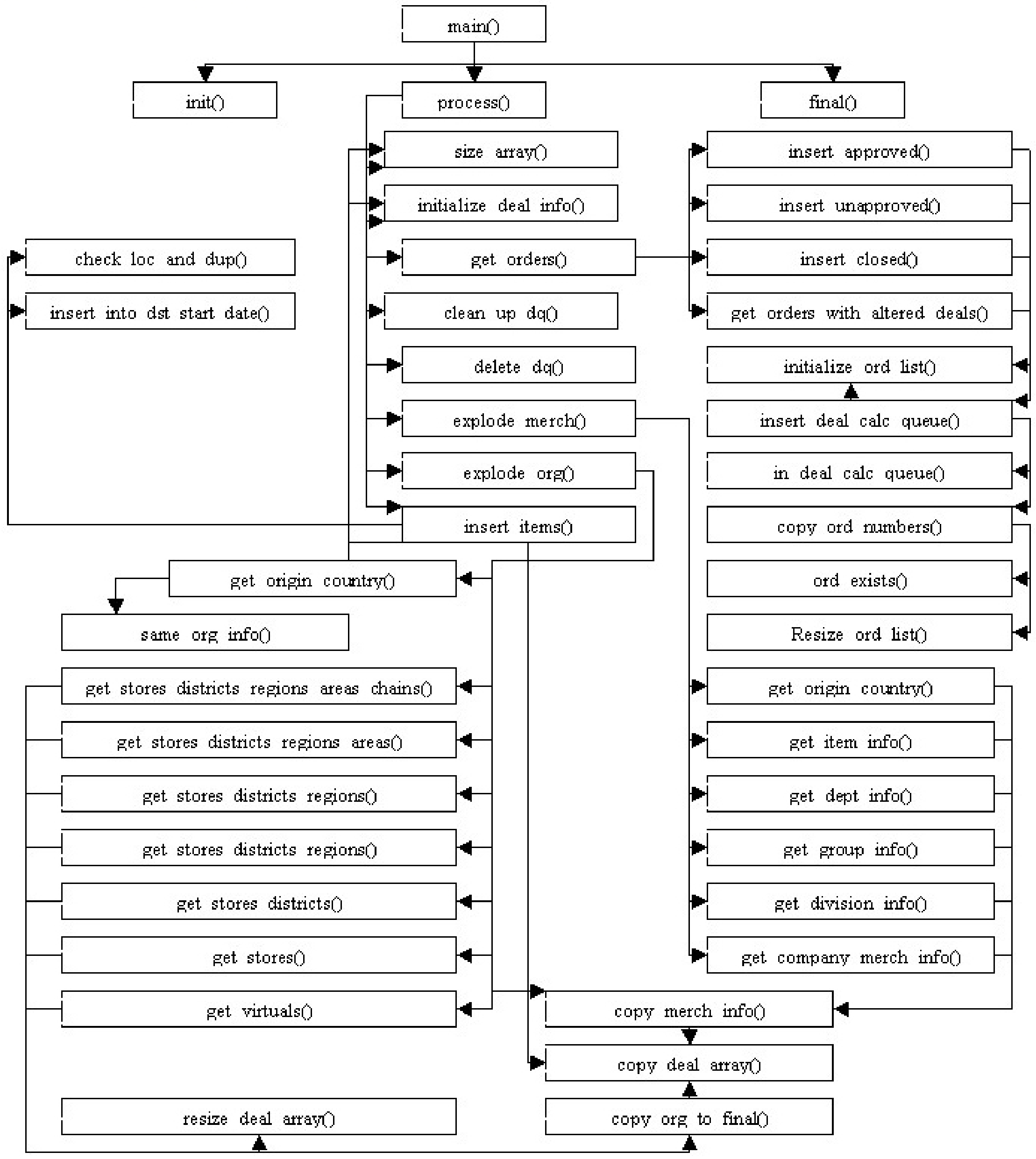
In addition, this program will populate the DEAL_CALC_QUEUE table with orders that may be affected by non vendor-funded, non PO-specific deals that are on the DEAL_QUEUE table (for future processing by orddscnt.pc). Orders that had been applied to deals that no longer apply will also be inserted into the DEAL_CALC_QUEUE table.

The LUW of this module is a single record from the DEAL_QUEUE table and deal definition information that belongs to the DEAL_ID from the DEAL_QUEUE table.

Stored Procedures / Shared Modules (Maintainability)

NONE

Program Flow



Function Level Description

Main(): Standard Retek main function. Validates input parameters, calls init, process and final. Logs appropriate message.

Init(): Standard Retek init function. Calls retek_init().

Process(): Drives the rest of the program:

- Call size_array() and initialize_deal_info().
- Array-fetch the driving cursor. For each fetched record:
    - If new deal, call delete_dq() and retek_force_commit().
    - If new deal, call get_orders() to populate DEAL_CALC_QUEUE.
    - Call explode_merch().
    - Call explode_org().
    - Call insert_items() to populate exploded merchandise and organizational records into DEAL_SKU_TEMP.
    - Save this deal id so next record's deal id can be compared to previous one.
- Call clean_up_dq().

Explode_merch():

- For each record from the deal_itemloc arrays**,** we need to obtain all missing merchandise hierarchy and origin country and supplier information, as follows:
- (Only vendor pack items should be used—skip buyer packs and non-approved, non-transaction level items.)
    - If merch_level = 12 get all origin countries for this item/supplier from the ITEM_SUPP_COUNTRY table if an origin country was not given. (We should already have all necessary merchandise hierarchy information). If no supplier was given, we must get all suppliers from the ITEM_SUPP_COUNTRY table that have the same partner as the deal (if the partner_id on deal_head is 'S1', get all suppliers from ITEM_SUPP_COUNTRY where supp_hier_lvl_1 matches the partner id; if it's 'S2', where supp_hier_lvl_2 matches, if 'S3', where supp_hier_lvl_3 matches. If the partner_id is 'S', a supplier is given on DEAL_HEAD; just match to that supplier. This supplier match must be done for all merch levels.
    - If merch_level = 11 (have info down to line/diff 4), need to get all items for the supplier/item parent/item grandparent/diff 4 from ITEM_MASTER and ITEM_SUPP_COUNTRY, and all origin countries from ITEM_SUPP_COUNTRY if the origin country was not given. (Remember that item parent, item grandparent, diff 1, diff 2, or diff_3 could be NULL).

- If merch_level = 10 (have info down to line/diff 3), need to get all items for the supplier/item parent/item grandparent/diff 3 from ITEM_MASTER and ITEM_SUPP_COUNTRY, and all origin countries from ITEM_SUPP_COUNTRY if the origin country was not given. (Remember that item parent, item grandparent, diff 1, or diff 2 could be NULL).
- If merch_level = 9 (have info down to line/diff 2), need to get all items for the supplier/item parent/item grandparent/diff 2 from ITEM_MASTER and ITEM_SUPP_COUNTRY, and all origin countries from ITEM_SUPP_COUNTRY if the origin country was not given. (Remember that item parent, item grandparent, diff 1, or diff 2 could be NULL).
- If merch_level = 8 (info down to line/diff 1), need to get all items (and diff 1 information) for this supplier/style/diff 1 from ITEM_MASTER and ITEM_SUPP_COUNTRY, and all origin countries (if the origin country was not given) from ITEM_SUPP_COUNTRY.
- If merch_level = 7 (info down to line), get all items for this supplier and line, along with all diff information and origin countries (if the origin country was not given) from ITEM_MASTER and ITEM_SUPP_COUNTRY.
- If merch_level = 6 (info down to subclass level), get all item and line/diff info for this supplier from ITEM_SUPP_COUNTRY, and ITEM_MASTER and all origin countries (if the origin country was not given) from ITEM_SUPP_COUNTRY.
- If merch_level = 5 (info down to class level), get all subclass/items/line/diff information info for this supplier from ITEM_SUPP_COUNTRY, and ITEM_MASTER and all origin countries (if the origin country was not given) from item_supp_country.
- If merch_level = 4 (info to dept level), get all class/subclass/item/line/diff information info for this supplier from ITEM_SUPP_COUNTRY, and ITEM_MASTER and all origin countries (if the origin country was not given) from item_supp_country.
- If merch_level = 3 (info to group level), get all departments for the group from DEPS, get classes/subclasses/items/lines/diffs for each department info for this supplier from ITEM_SUPP_COUNTRY, and ITEM_MASTER and all origin countries (if the origin country was not given) from item_supp_country.
- If merch_level = 2 (info to division level) get all groups under that division from GROUPS, and all departments under each group from DEPS. Then get classes/subclasses/items/lines/diffs for this supplier from ITEM_SUPP_COUNTRY, and ITEM_MASTER and all origin countries (if the origin country was not given) from item_supp_country.

- If merch_level = 1 (Company level--ALL items for a supplier), get all divisions from the DIVISION table, get all groups for each division from the GROUPS table, and all departments under each group from the DEPS table. Then get all items and their line and diff info for this supplier from ITEM_SUPP_COUNTRY, and ITEM_MASTER and all origin countries (if the origin country was not given) from ITEM_SUPP_COUNTRY.

Set up as a case statement; each merchandise level value calls a different function:

Case 10: Call get_origin_country().

Case 9, 8, or 7: Call get_item_info().

Case 6, 5, or 4: Call get_dept_info().

Case 3: Call get_group_info().

Case 2: Call get_division_info().

Case 1: Call get_company_merch_info().

Explode_org():

- For each record from the merchandise arrays (containing merchandise/country information and all organization information that was present on DEAL_ITEMLOC), get any additional organizational hierarchy information as follows:
    - If the org_level = 5 (location) and the given location is a warehouse, the chain, area, region, and district fields should be left as NULL when inserted into DEAL_SKU_TEMP–warehouses aren't part of a district (no additional information needs to be fetched). If multichannel is on, remember to blow out warehouse from DEAL_ITEMLOC (which always holds physical warehouses) to member virtual warehouses.
    - If org_level = 5 (location) and the given location is a store, all the necessary information should already be present; no additional information needs to be fetched.
    - If org_level = 4 (district), get all stores in this district from the store table. Again, the organizational hierarchy above the district should already have been copied from DEAL_ITEMLOC, so that does not need to be looked up.
    - If org_level = 3 (region), get all districts and stores in this region from the district and store tables. Again, the organizational hierarchy above the region should already have been copied from DEAL_ITEMLOC, so that does not need to be looked up.
    - If org_level =2 (area), get all regions, districts, stores for this area from region, district, and store tables. Again, the organizational hierarchy above the area should already have been copied from DEAL_ITEMLOC, so that does not need to be looked up.
    - If org_level = 1 (chain), get all areas, all regions, all districts, all stores for this chain from the area, region, district, and store tables.

- If nothing is given in the organizational hierarchy (org _level is null or 0), call get_stores_districts_regions_areas_chains() which will blow out the entire organizational hierarchy as it exists in RMS.

Keep track of how many organizational hierarchy combinations were fetched and for what level so that if the next record has the same organization level for the same organization hierarchy element (say, it's org level 4 and the current district is the same as the last district) the organization values can just be copied from the ones previously obtained instead of having to get them from the database again. This check is done by calling same_org_info().

As the information is obtained, it should be inserted into the final array (use copy_deal_array()) for insert to DEAL_SKU_TEMP (be sure to resize this array as necessary).

Set up as a case statement; each organizational level value calls a different function:

Case 5: If location is a store, or the location is a warehouse and the multichannel is turned off in the system, simply copy record, no lookup or blowout is needed. (Call copy_deal_array().) If the location is a warehouse and the multichannel option is turned on in the system, call get_virtuals() to blow out the physical warehouse from DEAL_ITEMLOC (which always only holds physical warehouses) to its virtual members.

Case 4: Call get_stores().

Case 3: Call get_stores_districts().

Case 2: Call get_stores_districts_regions().

Case 1: Call get_stores_districts_regions_areas().

Case 0: Call get_stores_districts_regions_areas_chains().

Same_org_info():

- This function will check if the previous record whose organizational hierarchy was blown out contains the same organizational level and parameters.

Get_origin_country():

- Given an array containing the records retrieved by the driving cursor, find all countries and suppliers for the given item, plus any line and diff information given, and copy the info into the passed output array. This function should only be called if all merchandise info is already known (merchandise level 12), except for the country.

Get_item_info():

- Given an array containing the records retrieved by the driving cursor, find all missing merchandise information, and copy the info into the passed output array. This function should only be called if some line and optionally diff information is given, but no item exists (merchandise levels 7-11).

Get_dept_info():

- Given an array containing the records retrieved by the driving cursor, find all missing merchandise information and copy the info into the passed output array. This function should only be called if dept is given, but line, diff and item is not given (merchandise levels 4-6).

Get_group_info():

- Given an array containing the records retrieved by the driving cursor, find all missing merchandise information and copy the info into the passed output array. This function should only be called if group number is given but dept and everything below is not given (merchandise level 3).

Get_division_info():

- Given an array containing the records retrieved by the driving cursor, find all missing merchandise information and copy the info into the passed output array. This function should only be called if division is given, but group number and everything below is not (merchandise level 2).

Get_company_merch_info():

- Given an array containing the records retrieved by the driving cursor, find all merchandise information and copy the info into the passed output array. This function should only be called if no merchandise information is given (merchandise level 1).

Get_stores_districts_regions_areas_chains():

- Given a specific deal in an array containing all items from the driving cursor with all merchandise information already filled in, find all locations and their organizational hierarchy existing in the company and copy them to the output array. Function also takes in another deal array and an indicator variable. If the indicator is true, then use organizational information contained in this third array rather than running queries.

Get_stores_districts_regions_areas():

- Given a specific deal in an array containing all items from the driving cursor with all merchandise information already filled in, find all locations and their organizational hierarchy existing under the given chain and copy them to the output array. Function also takes in another deal array and an indicator variable. If the indicator is true, then use org information contained in this third array rather than running queries.

Get_stores_districts_regions():

- Given a specific deal in an array containing all items from the driving cursor with all merchandise information already filled in, find all locations and their organizational hierarchy existing under the given area and copy them to the output array. Function also takes in another deal array and an indicator variable. If the indicator is true, then use org information contained in this third array rather than running queries.

Get_stores_districts():

- Given a specific deal in an array containing all items from the driving cursor with all merchandise information already filled in, find all locations and their organizational hierarchy existing under the given region and copy them to the output array. Function also takes in another deal array and an indicator variable. If the indicator is true, then use org information contained in this third array rather than running queries.

Get_stores():

- Given a specific deal in an array containing all items from the driving cursor with all merchandise information already filled in, find all locations and their organizational hierarchy existing under the given district and copy them to the output array. Function also takes in another deal array and an indicator variable. If the indicator is true, then use org information contained in this third array rather than running queries.

Get_virtuals():

- Given a specific deal in an array containing all items from the driving cursor with all merchandise information already filled in, find all locations existing under the given district and copy them to the output array. (Note that warehouses have no organizational hierarchy.) Function also takes in another deal array and an indicator variable. If the indicator is true, then use org information contained in this third array rather than running queries.

Copy_merch_info():

- This function takes three deal record arrays and an index. The first array contains the information gathered from the driving cursor. The second contains the information gathered from the preceding get_<level>_info functions. For each record in the second array, copy all merchandise related information from the second array to the first array. Then insert the combined record into the third array using copy_deal_array().

Initialize_deal_info():

- Initializes all indicator variables in every record and sets deal record count to 0 for a passed deal struct.

Insert_items():

- Array insert records into the DEAL_SKU_TEMP table, first checking for duplicates (check to make sure that each record does not yet exist in DEAL_SKU_TEMP and that the item-location combinations exist in ITEM_LOC; if check was OK, put it into an array for insert; insert once all have been checked).

To perform the above task:

- Call necessary sizing and initialization functions to allocate final insert array. (Call size_array(), initialize_array().)
- For each record call check_loc_and_dup().
- If previous function call indicated that the record is good to go for the insert, call copy_deal_array() to insert that record into the final insert array.
- Check to see if the current record has a reset date. If it does, copy that reset date into the active date and call check_loc_and_dup() again. If current record is ok to insert, call copy_deal_array() to copy this record into the final insert array. This way we insert the deal's close date for its items into DEAL_SKU_TEMP.
- When records are checked and the final insert array's size is greater than 0, call insert_into_dst_start_date() to array insert the final records.

Check_loc_and_dup():

- Check to make sure insert record does not exist in DEAL_SKU_TEMP and check to make sure insert record's item-location relationship does exist in ITEM_LOC. When checking the item-location relationship from the ITEM_LOC table, also verify that the item is not the component of a primary costing pack. Such records are not inserted, only the component's case UPC.

Insert_into_dst_start_date():

- Insert passed record into DEAL_SKU_TEMP.

Ord_exists():

- Sets return_flag to true if passed ord is already in passed ord_list.

Copy_ord_to_final():

- This function takes an output array, an array containing all merch information, and an array containing org information. For each record in the org array, copy information to given record in the merch array, and place this combined record into the output array.

Copy_ord_numbers():

- Given to ord_lists and an index, copy value at index from start array to final array if the value does not already exists in the final array.

In_deal_calc_queue():

- Set return flag to true if passed ord_num is already in DEAL_CALC_QUEUE.

Insert_deal_calc_queue():

- Insert values contained in given ord_list into DEAL_CALC_QUEUE.

Size_ord_list():

- Allocate memory for ord_list, and set ord_list count to 0.

Resize_ord_list():

- Allocate additional memory for deal array. Allocation is incremented by commit_max_ctr.

Get_orders():

- Given a deal id, a deal status, the deal's recalculate approved orders indicator, and the deal start (active) and close dates, do the following:
    - If the deal's recalculate approved orders indicator is 'Y' and the status is 'A'pproved, call insert_approved().
    - Call insert_unapproved() if the status is 'W'orksheet (it was unapproved and just got set back to worksheet status).
    - Call insert_closed() if the status is 'C'losed.
    - Call get_orders_with_altered_deals() if the status is 'A'pproved. (This is done for approved deals that had no close date and just got closed, which could disqualify some orders that have already been calculated with this deal from using this deal.)

This will populate the DEAL_CALC_QUEUE table with orders that may be affected by the deal.

Get_orders_with_altered_deals():

- Select all approved orders from the ORDHEAD and ORDLOC_DISCOUNT tables whose not before date is higher than the deal close date. Also make sure the contract_no field on ORDHEAD is null. Insert these orders into the DEAL_CALC_QUEUE table (recalc_all_ind, order_appr_ind , and override_manual_ind should be inserted as 'N').

Insert_approved():

- Select all approved orders from the ORDHEAD table whose not before date is between the deal start date and the deal close date (or just after the deal start date, if the deal close date is NULL). Also make sure the contract_no field on ORDHEAD is null. Insert these orders into the DEAL_CALC_QUEUE table (recalc_all_ind, order_appr_ind , and override_manual_ind should be inserted as 'N').

Insert_unapproved():

- Select all orders from the ORDLOC_DISCOUNT table for the given deal that do not have a status (on ORDHEAD) of 'C'losed, and insert these orders into the DEAL_CALC_QUEUE table (recalc_all_ind, order_appr_ind , and override_manual_ind should be inserted as 'N'). Also make sure the contract_no field on ORDHEAD is null.

Insert_closed():

- Select all orders from the ORDLOC_DISCOUNT table for the given deal whose not before date is after the deal's close date (all closed deals must have a close date) and whose status (from ORDHEAD) is not 'C'losed. Also make sure the contract_no field on ORDHEAD is null. Insert these orders into the DEAL_CALC_QUEUE table (recalc_all_ind, order_appr_ind , and override_manual_ind should be inserted as 'N').

Size_array():

- Allocate memory for passed array.

Resize_deal_array():

- Allocate additional memory for deal array. Allocation is incremented by commit_max_ctr.

Copy_deal_array():

- Given two deal arrays and an index, copy deal at index from I_deal_info to o_unique_records. This function will resize o_unique_records if necessary, but it will not check if deal already exists in the final array.

Delete_dq():

- Delete processed records from the DEAL_QUEUE table.

Clean_up_dq():

- Delete all records left on the DEAL_QUEUE table.

Final():

- Performs restart/recovery close logic. Calls retek_close().

Input Specifications

'Table-To-Table'

Select data from:

Table Name	Column Name	Column Type	Transformation
DEAL_HEAD	SUPPLIER	NUMBER(10)	NONE
DEAL_HEAD	PARTNER_TYPE	VARCHAR2(6)	NONE
DEAL_HEAD	PARTNER_ID	VARCHAR2(10)	NONE
DEAL_HEAD	ACTIVE_DATE	DATE	NONE
DEAL_HEAD	CLOSE_DATE	DATE	NONE
DEAL_HEAD	STATUS	VARCHAR2(1)	NONE
DEAL_HEAD	RECALC_APPROVED_ORDERS	VARCHAR2(1)	NONE
DEAL_QUEUE	DEAL_ID	NUMBER(10)	NONE
DEAL_ITEMLOC	DEAL_DETAIL_ID	NUMBER(10)	NONE
ITEM_MASTER	ITEM	VARCHAR2(25)	NONE
ITEM_MASTER	ITEM_PARENT	VARCHAR2(25)	NONE
ITEM_MASTER	ITEM_GRANDPARENT	VARCHAR2(25)	NONE
ITEM_MASTER	DIFF_1	VARCHAR2(10)	NONE

Table Name	Column Name	Column Type	Transformation
ITEM_MASTER	DIFF_2	VARCHAR2(10)	NONE
ITEM_MASTER	DIFF_3	VARCHAR2(10)	NONE
ITEM_MASTER	DIFF_4	VARCHAR2(10)	NONE
ITEM_MASTER	SUBCLASS	NUMBER(4)	NONE
ITEM_MASTER	CLASS	NUMBER(4)	NONE
DEPS	DEPT	NUMBER(4)	NONE
GROUPS	GROUP_NO	NUMBER(4)	NONE
DIVISION	DIVISION	NUMBER(4)	NONE
STORE	STORE	NUMBER(4)	NONE
DISTRICT	DISTRICT	NUMBER(4)	NONE
REGION	REGION	NUMBER(4)	NONE
AREA	AREA	NUMBER(4)	NONE
CHAIN	CHAIN	NUMBER(4)	NONE
ORDHEAD	ORDER_NO	NUMBER(8)	NONE
ORDLOC_DISCOUNT	ORDER_NO	NUMBER(8)	NONE
SYSTEM_OPTIONS	MULTICHANNEL_IND	VARCHAR2(1)	NONE

Output Specifications

‘Table-To-Table’

Delete data from

Table Name	Column Name	Column Type	Transformation
DEAL_QUEUE	DEAL_ID	NUMBER(10)	N/A

The following table will be inserted:

Table Name	Column Name	Column Type	Transformation
DEAL_SKU_TEMP	ITEM	VARCHAR2(25)	NONE
DEAL_SKU_TEMP	SUPPLIER	NUMBER(10)	NONE
DEAL_SKU_TEMP	ORIGIN_COUNTRY_ID	VARCHAR2(3)	NONE
DEAL_SKU_TEMP	START_DATE	DATE	NONE
DEAL_SKU_TEMP	DIVISION	NUMBER(4)	NONE
DEAL_SKU_TEMP	GROUP_NO	NUMBER(4)	NONE
DEAL_SKU_TEMP	DEPT	NUMBER(4)	NONE

Table Name	Column Name	Column Type	Transformation
DEAL_SKU_TEMP	CLASS	NUMBER(4)	NONE
DEAL_SKU_TEMP	SUBCLASS	NUMBER(4)	NONE
DEAL_SKU_TEMP	ITEM_PARENT	VARCHAR2(25)	NONE
DEAL_SKU_TEMP	ITEM_GRANDPARENT	VARCHAR2(25)	NONE
DEAL_SKU_TEMP	DIFF_1	VARCHAR2(10)	NONE
DEAL_SKU_TEMP	DIFF_2	VARCHAR2(10)	NONE
DEAL_SKU_TEMP	DIFF_3	VARCHAR2(10)	NONE
DEAL_SKU_TEMP	DIFF_4	VARCHAR2(10)	NONE
DEAL_SKU_TEMP	CHAIN	NUMBER(4)	NONE
DEAL_SKU_TEMP	AREA	NUMBER(4)	NONE
DEAL_SKU_TEMP	REGION	NUMBER(4)	NONE
DEAL_SKU_TEMP	DISTRICT	NUMBER(4)	NONE
DEAL_SKU_TEMP	LOCATION	NUMBER(10)	NONE
DEAL_SKU_TEMP	LOC_TYPE	VARCHAR2(1)	NONE
DEAL_CALC_QUEUE	ORDER_NO	NUMBER(8)	NONE
DEAL_CALC_QUEUE	RECALC_ALL_IND	VARCHAR2(1)	Will always be 'N'.
DEAL_CALC_QUEUE	OVERRIDE_MANUAL_IND	VARCHAR2(1)	Will always be 'N'.
DEAL_CALC_QUEUE	ORDER_APPR_IND	VARCHAR2(1)	Will always be 'N'.

Scheduling Considerations

This program should run as the first batch program in the deals batch cycle.

Locking Strategy

N/A

Restart/Recovery

The module has restart/recovery built in based on DEAL_ID from the DEAL_QUEUE table.

Performance Considerations

N/A

Security Considerations

N/A

Design Assumptions

Primary cost pack component items do not get inserted into DEAL_SKU_TEMP, only their case UPC items. An item must be in ‘A’pproved status and at the transaction level, it must not be a buyer pack. Orders may have no contracts in order for them to be inserted into DEAL_CALC_QUEUE.

Outstanding Design Issues

N/A

EDI contract information download [edidlcon]

Design Overview

This program downloads a file of EDI contract information. Contracts are only processed if they are in approved status and have an edi_contract_ind of ‘Y’.

Changes to make: add restart recovery and make output ONE file instead of one per supplier. Minor changes to file format (add Gentran ID in FHEAD line and move supplier from FHEAD to THEAD; include transaction number on transaction lines). Let the user enter an output file name.

Scheduling Constraints

Processing Cycle:	Daily, Phase 4
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A –file based processing

Restart Recovery

```
SELECT ch.contract_no,
       ch.contract_type,
       ch.dept,
       TO_CHAR(ch.supplier),
       TO_CHAR(NVL(ch.total_cost*1000,0)),
       ch.edi_contract_ind,
          ch.currency_code,
          ROWIDTOCHAR(ch.rowid)
  FROM contract_header ch
 WHERE ch.status = ’A’
   AND ch.edi_contract_ind = ‘Y’
   AND ch.edi_sent_ind = ‘N’
 ORDER BY ch.supplier
```

Program Flow

N/A

Shared Modules

CONTRACT_SQL.GET_UNIT_COST – get the cost of a contract-item.

Function Level Description

Include the std_rest.h library.

Init:

Get period.vdate.

Call restart_file_init

Make the format strings for output file lines

Open output file and write fhead line with the write_head function

Process:

Fetch contract header cursor

Update contract_header.edi_sent_ind to 'Y'

write information out to file (call write_detail and write_summary)

use restart_commit for commits

write_head

write FHEAD line to file

write_detail

For contract types C and D (no production plans): Get item contract info from the contract_detail table. Get ref_item and ref_item type info from the item_master table. Write TDETL lines to file. For contract types A and B (production plans): Get item contract info from the contract_detail table. Get ref_item info from the item_master table. Write TDETL lines to file. (Note: ready_date, ready_quantity, location_type, and location_number will only have values for contract types A and B. Furthermore, in a multi-channel environment, all TDETL records need to be written at the physical location level.)

```
write_summary:

write TTAIL string to file


write_trailer

write FTAIL string to file


 Final:

Write output to final file and close files.

Call restart_close.
```

I/O Specification

Output File layout:

FHEAD File identification and date

THEAD Supplier and contract header info

TDETL Item information

TTAIL Transaction trailer

FTAIL File trailer; total number of transactions written

All character variables should be right-padded with blanks and left_justified; all numerical variables should be left-padded with zeroes and right-justified. All dates should be in YYYYMMDDHH24MISS format.

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record descriptor	Char(5)	FHEAD	Describes file line type
	Line number	Number(10)	0000000001	Sequential file line number
	Gentran ID	Char(5)	DNCN	Identifies transaction type for Gentran
	Current date	Char(14)		Current date period.vdate
THEAD	Record descriptor	Char(5)	THEAD	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Supplier	Char(10)		Contract_header.supplier
	Contract	No number (6)		Contract_header.contract_no
	Contract type	Char(1)		Contract_header.contract_typ e
	Department	Number(4)		Contract_header.dept
	Currency code	Char(3)		Contract_header.currency_co de
	Total contract cost	Char(20)		Contract_header.total-cost*10000
TDETL	Record descriptor	Char(5)	TDETL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number

Record Name	Field Name	Field Type	Default Value	Description
	Item Number Type	Char(6)		Item type for item from item_master table.
	Item Number	Char(25)		item
	Ref Item Number Type	Char(6)		Reference item number type retrieved from item_master.item_number_type.
	Ref Item number	Char(25)		Primary reference item retrieved from Item Master table.
	Diff1 Desc	Char(40)		Diff 1 Description
	Diff2 Desc	Char(40)		Diff 2 Description
	Diff3 Desc	Char(40)		Diff 3 Description
	Diff4 Desc	Char(40)		Diff 4 Description
	VPN	Char(30)		Vendor Product Number for an item
	Unit cost	Char(2)		Contract_sku.unit_cost*10000 (4 implied decimal places)
The following variables will only have values for contract types of ‘A’ or ‘B’;				
	Ready date	Char(14)		Contract_prod_plan.ready_date
	Ready quantity	Char(20)		Contract_prod_plan.qty_ready*10000 (4 implied decimal places)
	Location type	Char(2)		‘ST’ or ‘WH’
	Location number	Char(10)		Contract_prod_plan.store or .wh
FTAIL	Record descriptor	Char(5)	TTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
FTAIL	Record descriptor	Char(5)	FTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number (total # lines in file)
	Contract count	Number(10)		Total number of transactions in file

Technical Issues

N/A

EDI purchase order download [edidlord]

Functional Area

Purchase Orders

Design Overview

Orders generated within the Retek system are written to a flat file if they are approved and specified as EDI orders. If shipments are to be pre-marked for cross-dock allocation by the supplier, allocation location and quantities will be sent along with the order information. If the order contains pack items, hierarchical pack information will be sent (this may include outer packs, inner packs, and fashion styles with associated pack templates, as well as component item information). File output is to a Retek standard format file, with the translation to EDI format taking place via an outside translator such as Gentran.

In the past, edidlnew downloaded new orders to an output file, while edidlchg downloaded changed orders. These programs were combined and modified to work with changes that have been made to the ordering tables. The order revision tables and allocation revision table will also be used, to ensure that the latest changes are being sent and to allow both original and modified values to be sent. These revision tables are populated during the online ordering process and the batch replenishment process whenever an order has been approved, and constitute a history of all revisions to the order.

If multi-channel is turned on in the system, the program will sum all quantities to the physical warehouse level for an order before writing the output file.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ORDHEAD_REV	Yes	Yes	No	No	No
ORDHEAD	Yes	No	No	Yes	No
ORDSKU	Yes	Yes	No	No	No
ORDLOC	Yes	Yes	No	No	No
ORDSKU_REV	Yes	Yes	No	No	No
ORDLOC_REV	Yes	Yes	No	No	No
ITEM_SUPPLIER	No	Yes	No	No	No
ITEM_MASTER	Yes	Yes	No	No	No
WH	Yes	Yes	No	No	No
ALLOC_HEADER	Yes	No	No	No	No
ALLOC_DETAIL		Yes	No	No	No
ALLOC_DETAIL_REV		Yes	No	No	No
DESC_LOOK		Yes	No	No	No
PACKITEM_BREAKOUT		Yes	No	No	No

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
SUPS_PACK_TEMPL_DESC		Yes	No	No	No

Stored Procedures / Shared Modules (Maintainability)

ELC_CALC_SQL.CALC_BACKHAUL_TOTAL – Calculates the backhaul allowance for an order.

Program Flow

Orders that are in approved status and that are new or have had changes made are fetched from the system tables. Additional information about the items on the order and their destination is gathered. Order, item, pack, and shipment information is written to an output file. The system tables are then updated to show that the orders have been sent.

```
main()
  |
  +-init()
  |  |
  |  +-init_terms_array()
  |
  +-process()
  |  |
  |  +-LOOP for each p.o.
  |  |  |
  |  |  +-if backhaul_type = C then
  |  |  |  |
  |  |  |  +-calc_backhaul()
  |  |  |     |
  |  |  |     +-ELC_CALC_SQL.CALC_BACKHAUL_TOTAL()
  |  |  |
  |  |  +-get_terms_des()
  |  |  |
  |  |  +-write_TORDR
  |  |  |  |
  |  |  |  +print TORDR to output file
  |  |  |
  |  |  +-write_items
  |  |  |  |
  |  |  |  +LOOP for each item in p.o.
  |  |  |  |  |
  |  |  |  |  +-get_item_type()
  |  |  |  |  |
  |  |  |  |  +-get_supp_item()
  |  |  |  |  |
  |  |  |  |  +-get_ref_item()
  |  |  |  |  |
  |  |  |  |  +-write_TITEM()
  |  |  |  |  |  |
  |  |  |  |  |  +print TITEM to output file
  |  |  |  |  |
  |  |  |  |  +-get_pack()
  |  |  |  |  |  |
  |  |  |  |  |  +LOOP for each item in pack
  |  |  |  |  |  |  |
  |  |  |  |  |  |  +get_supp_item()
  |  |  |  |  |  |  |
  |  |  |  |  |  |  +get_ref_item()
  |  |  |  |  |  |  |
  |  |  |  |  |  |  +write_TPACK()
  |  |  |  |  |  |  |  |
  |  |  |  |  |  |  |  +print TPACK to output file
  |  |  |  |  |  |  |
  |  |  |  |  |  |--< end LOOP
  |  |  |  |  |
  |  |  |  |  +-write_shipto()
  |  |  |  |  |  |
  |  |  |  |  |  +LOOP for each item/location on p.o.
  |  |  |  |  |  |  |
  |  |  |  |  |  |  +get_item_dims()
  |  |  |  |  |  |  |
  |  |  |  |  |  |  +write_TSHIP()
  |  |  |  |  |  |  |  |
  |  |  |  |  |  |  |  +print TSHIP to output file
  |  |  |  |  |  |  |
  |  |  |  |  |  |--< end LOOP
  |  |  |  |  |
  |  |  |  |  +-write_alloc()
  |  |  |  |  |  |
  |  |  |  |  |  +LOOP for each allocation record on p.o.
  |  |  |  |  |  |  |
  |  |  |  |  |  |  +get_item_dims()
  |  |  |  |  |  |  |
  |  |  |  |  |  |  +write_TSHIP()
  |  |  |  |  |  |  |  |
  |  |  |  |  |  |  |  +print TSHIP to output file
  |  |  |  |  |  |  |
  |  |  |  |  |  |--< end LOOP
  |  |  |  |  |
  |  |  |  |--< end loop
  |  |  |
  |  |  +-write_TTAIL
  |  |  |  |
  |  |  |  +print TTAIL to output file
  |  |  |
  |  |  +-update ordhead
  |  |  |
  |  |  +-restart_commit()
  |  |  |
  |  |  +-restart_file_write()
  |  |  |
  |  |--< end loop
  |
  +-final()
     |
     +print FTAIL to output file
```

Function Level Description

init()

Get current date. Set up format strings for output file, open output file, and write file header. Set up restart/recovery. Call init_terms_array to fetch terms and descriptions from terms table.

process()

Select the new or changed orders in approved status for EDI download by fetching the driving cursor. Eligible orders are approved and have an EDI indicator set. The cursor selects “new” values from ordhead and “old” values from ordhead_rev for the next to last version (the last version is the current one, with the same information that is now on ordhead). For a new order, no earlier version will exist on the ordhead_rev table, so no “old” values will be fetched. If old values are null, this must mean that we have a new order. Information from different suppliers can be sent in the same file, but the file will be sorted by supplier. If the backhaul type = C (calculated) then call calc_backhaul function to calculate the backhaul totals. Call get_terms_des to fetch the description of the terms code. Call write_TORDR to write order header level information to file. Call write_items to fetch additional item-level information and write it to output file. Update the ordhead table to show that an EDI transaction has been sent and an acknowledgment has not yet been received.

write_TORDR()

Write the TORDR line (order header level information) to the output file.

write_items()

Get item information (from the ordsku and ordsku_rev tables--quantity ordered, outstanding quantity, description). Get item cost information and supplier information (by calling get_supp_item). If reference item information does not exist on ordsku, call get_ref_item to fetch the ref item information (if any). Call write_TITEM to write item information line to file. If the item is a pack identifier (pack_ind =’Y’ on item_master), call get_pack to get information on component items within the pack. If the order is to be pre-marked, call write_alloc to write allocation information to file; otherwise write shipment information to file by calling write_shipto.

write_TITEM()

Write item information line to file.

write_shipto()

Fetch shipment location and quantity information for a particular item on the order from the ordloc and ordloc_rev tables. Call get_item_dims to get the case dimensions then call write_TSHIP to write it out to output file.

write_alloc()

This function is called only for cross-docked allocations that will be pre-marked by the supplier. Fetch allocation information from the alloc_header, alloc_detail, and alloc_rev tables, call get_item_dims, then call write_TSHIP to write it to output file.

get_pack()

Get information on items contained within a pack (from the packitem_breakout table). If the item is part of a pack template, fetch the template description from the supps_pack_tmpl_desc table. Call get_supp_item() to get the item_supplier for each of the items. Use get_ref_item to fetch ref item information for these items. Call write_TPACK to write pack item information lines to file.

write_TPACK()

Write pack component information lines to file.

get_supp_item()

Get supplier VPN, supplier's color code and supplier's size code from the item_supplier table.

get_ref_item()

Get ref item info (either primary or preferred for supplier)

write_TSHIP()

Write TSHIP line to file—shipment location and quantity info. Also used to write allocation information

write_TTAIL()

Write order trailer line to file.

calc_backhaul()

Call ELC_CALC_SQL.CALC_BACKHAUL_TOTAL to get the backhaul allowance for this order.

init_terms-array()

Fetches all terms and descriptions from terms table so the terms table doesn't have to be joined for each TORDR record.

get_terms_des()

Searches the terms array for a desciption.

get_item_dims()

Gets case dimensions from item_supp_country_dim.

final()

Write file trailer, copy temporary file to final file (restart/recovery close), close files.

Input Specifications

Command Line Parameters:

edidlord userid/password input_file

For a new order, the "old" fields should be blank. For a changed order, both old and new fields should hold values, if value has changed. "Old" values come from the revision tables for the latest revision before the current one (the last one sent), while new orders come from the ordering tables.

FHEAD – REQUIRED. File identification, one line per file.
TORDR – REQUIRED. Order level info, one line per order.

TITEM – REQUIRED. Item description, multiple lines per order possible.

TPACK – OPTIONAL. Pack contents, multiple lines per order possible. This line will be written only for pack items.

TSHIP – REQUIRED. Ship to location and quantity, allocation location, multiple lines per item possible. Allocation information is optional on this line—will exist if premark_ind is 'Y'.

TTAIL – REQUIRED. Order end, one line per order.

FTAIL – REQUIRED. End of file marker, one line per file.

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	**Record descriptor**	**Char(5)**	**FHEAD**	**File head marker**
	Line id	Char(10)	0000000001	Unique line id
	Translator id	Char(5)	DLORD	Identifies transaction type
	File create date	Char(14)	Current date	YYYYMMDDHH24MISS format
TORDR	**Record descriptor**	**Char(5)**	**TORDR**	**Order header info**
	Line id	Char(10)		Unique file line id
	Transaction id	Char(10)		Unique transaction id
	Order change type	Char(2)		'CH' (changed) or 'NW' (new)
	Order number	Number(8)		Internal Retek order no
	Supplier	Number(10)		Internal Retek supplier id
	Vendor order id	Char(15)		External vendor_order_no (if available)
	Old order written date	Char(14)		Old date order created YYYYMMDDHH24MISS
	New order written date	Char(14)		Changed date order created YYYYMMDDHH24MISS
	Old Currency Code	Char(3)		Old order currency_code (ISO standard)
	New Currency Code	Char(3)		Changed order currency_code (ISO standard)
	Old Shipment Method of payment	Char(2)		Old ship_pay_method
	New Shipment Method of Payment	Char(2)		Changed ship_pay_method
	Old Transportation Responsibility	Char(2)		Old fob_trans_res

Record Name	Field Name	Field Type	Default Value	Description
	New Transportation Responsibility	Char(2)		Changed fob_trans_res
	Old Trans. Resp. Description	Char(45)		Old fob_trans_res_desc
	New Trans. Resp. Description	Char(45)		New fob_trans_res_desc
	Old Title Passage Location	Char(2)		Old fob_title_pass
	New Title Passage Location	Char(2)		Changed fob_title_pass
	Old Title Passage Description	Char(45)		Old fob_title_pass_desc
	New Title Passage Description	Char(45)		Changed fob_title_pass_desc
	Old not before date	Char(14)		Old not_before_date YYYYMMDDHH24MISS
	New not before date	Char(14)		Changed not_before_date YYYYMMDDHH24MISS
	Old not after date	Char(14)		Old not_after_date YYYYMMDDHH24MISS
	New not after date	Char(14)		Changed not_after_date YYYYMMDDHH24MISS
	Old Purchase type	Char(6)		Old Purchase type
	New Purchase type	Char(6)		New Purchase type
	Backhaul allowance	Number(20)		Backhaul allowance
	Old terms description	Char(240)		Old terms description from terms table
	New terms description	Char(240)		New terms description from terms table
	Old pickup date	Char(14)		Old pickup date YYYYMMDDHH24MISS
	New pickup date	Char(14)		New pickup date YYYYMMDDHH24MISS
	Old ship method	Char(6)		Old ship method
	New ship method	Char(6)		New ship method

Record Name	Field Name	Field Type	Default Value	Description
	Old comment description	Char(250)		Old comment description
	New comment description	Char(250)		New comment description
	Supplier DUNS number	Number(9)		Supplier DUNS number
	Supplier DUNS location	Number(4)		Supplier DUNS location
	File record descriptor	**Char(5)**		**Item info**
	Line id	Char(10)		Unique line id
	Transaction id	Char(10)		Unique transaction id
	Item Number Type	Char(6)		Item_number_type
	Item	Char(25)		Item (If a pack item, this will be the pack number)
	Old Ref Item Number type	Char(6)		Item_number_type for old ref_item
	Old Ref Item	Char(25)		Old Ref_Item
	New Ref Item Number type	Char(6)		Item_number_type for new ref_item
	New Ref Item	Char(25)		Changed Ref_Item
	Vendor catalog number	Char(30)		Supplier_item (VPN)
	Free Form Description	Char(100)		item_desc
	Supplier Diff 1	Char(80)		Supplier’s diff 1
	Supplier Diff 2	Char(80)		Supplier’s diff 2
	Supplier Diff 3	Char(80)		Supplier’s diff 3
	Supplier Diff 4	Char(80)		Supplier’s diff 4
	Pack Size	Number(12)		Supplier defined pack size
TPACK	**File record descriptor**		**TPACK**	**Pack component info**
	Line id	Char(10)		Unique line id
	Transaction id	Char(10)		Unique transaction id
	Pack id	Char(25)		Packitem_breakout.pack_no (same as item for the pack item)
	Inner pack id	Char(25)		Inner pack identification

Record Name	Field Name	Field Type	Default Value	Description
	Pack Quantity	Number(12)		Packitem_breakout.pack_item_qty (4 implied decimal places)
	Component Pack Quantity	Number(12)		Packitem_breakout.comp_pack_qty (4 implied decimal places)
	Item Parent Part Quantity	Number(12)		Packitem_breakout.item_parent_pt_qty (4 implied decimal places)
	Item Quantity	Number(12)		Packitem_breakout.item_qty (4 implied decimal places)
	Item Number Type	Char(6)		Item number type
	Item	Char(25)		Item
	Ref Item Number Type	Char(6)		Ref_item_number_type
	Ref Item	Char(25)		Ref_item
	VPN	Char(30)		Supplier item (vpn)
	Supplier Diff 1	Char(80)		Supplier’s diff 1
	Supplier Diff 2	Char(80)		Supplier’s diff 2
	Supplier Diff 3	Char(80)		Supplier’s diff 3
	Supplier Diff 4	Char(80)		Supplier’s diff 4
	Item Parent	Char(25)		Required when Pack Template is not NULL
	Pack template	Char(8)		Pack template associated w/style (packitem_breakout.pack_tmpl_id)
	Template description	Char(40)		Description of pack template (if present) sups_pack_tmpl_desc.supp_pack_desc
TSHIP	**Record type**	**Char(5)**	**TSHIP**	**Describes file record- shipment info**
	Line id	Char(10)		Unique file line number
	Transaction id	Char(10)		Unique transaction number
	Location type	Char(2)		‘ST’ store or ‘WH’ warehouse
	Ship to location	Number(10)		Location value form ordloc (store or wh)
	Old unit cost	Number(20)		Old unit cost (4 implied decimal places)

Record Name	Field Name	Field Type	Default Value	Description
	New unit cost	Number(20)		New unit cost (4 implied decimal places)
	Old quantity	Number(12)		Old qty_ordered or qty_allocated (4 implied decimal places)
	New quantity	Number(12)		Changed qty_ordered or qty_allocated (4 implied decimal places)
	Old outstanding quantity	Number(12)		Old qty_ordered-qty_received (4 implied decimal places)(or qty_allocated-qty transferred, for an allocation)
	New outstanding quantity	Number(12)		Changed qty_ordered-qty_received (4 implied decimal places)(or qty_allocated-qty_transferred, for an allocation)
	Cancel code	Char(1)		
	Old cancelled quantity	Number(12)		Previous quantity cancelled (4 implied decimal places)
	New cancelled quantity	Number(12)		Changed quantity cancelled (4 implied decimal places)
	Quantity type flag	Char(1)		‘S’hip to ‘A’llocate
	Store or warehouse indicator	Char(2)		‘ST’ (store) or ‘WH’ (warehouse)
	Old x-dock location	Number(10)		Alloc_detail location (store or wh)
	New x-dock location	Number(10)		Alloc_detail location (store or wh)
	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)
	Case LWH unit of measure	Char(4)		Case LWH unit of measure
	Case weight	Number(12)		Case weight (4 implied decimal places)

Record Name	Field Name	Field Type	Default Value	Description
	Case weight unit of measure	Char(4)		Case weight unit of measure
	Case liquid volume	Number(12)		Case liquid volume (4 implied decimal places)
	Case liquid volume unit of measure	Char(4)		Case liquid volume unit of measure
	Location DUNS number	Number(9)		Location DUNS number
	Location DUNS loc	Number(4)		Location DUNS loc
	New unit cost init	Number(20)		New unit cost init (4 implied decimal places)
	Old unit cost init	Number(20)		Old unit cost init (4 implied decimal places)
	Item/loc discounts	Number(20)		Item/loc discounts (4 implied decimal places)
TTAIL	**Record type**	**Char(5)**	**TTAIL**	**Describes file record – marks end of order**
	Line id	Char(10)		Unique file line id
	Transaction id	Char(10)		Unique transaction id
	#lines in transaction	Number(10)		#lines in transaction
FTAIL	**Record type**	**Char(5)**	**FTAIL**	**Describes file record – marks end of file**
	Line id	Char(10)		Unique file line id
	#lines	Number(10)		Total number of transaction lines in file (not including FHEAD and FTAIL)

Output Specifications

N/A

Scheduling Considerations

Processing Cycle: PHASE 4 (may also be schedule ad hoc to run multiple times per day)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Locking Strategy

N/A

Restart/Recovery

```
Driving cursor:

   SELECT ROWIDTOCHAR(oh.rowid),
          oh.order_no,
          to_char(oh.supplier),
          to_char(oh.written_date,'YYYYMMDDHH24MISS'),
          to_char(ohr.written_date,'YYYYMMDDHH24MISS'),
          to_char(ohr.not_before_date,'YYYYMMDDHH24MISS'),
          to_char(oh.not_before_date,'YYYYMMDDHH24MISS'),
          to_char(ohr.not_after_date,'YYYYMMDDHH24MISS'),
          to_char(oh.not_after_date,'YYYYMMDDHH24MISS'),
          oh.vendor_order_no,
          ohr.currency_code,
          oh.currency_code,
          ohr.ship_pay_method,
          oh.ship_pay_method,
          ohr.fob_trans_res,
          oh.fob_trans_res,
          ohr.fob_trans_res_desc,
          oh.fob_trans_res_desc,
          ohr.fob_title_pass,
          oh.fob_title_pass,
          ohr.fob_title_pass_desc,
          oh.fob_title_pass_desc,
          oh.pre_mark_ind,
          oh.last_sent_rev_no,
          ohr.purchase_type,
          oh.purchase_type,
          oh.backhaul_type,
          NVL(oh.backhaul_allowance,0) * POWER(10,:pi_qty_dec),
          oh.exchange_rate,
          ohr.terms,
          oh.terms,
          to_char(ohr.pickup_date,'YYYYMMDDHH24MISS'),
          to_char(oh.pickup_date,'YYYYMMDDHH24MISS'),
          ohr.ship_method,
          oh.ship_method,
          ohr.comment_desc,
          oh.comment_desc,
          s.duns_number,
          s.duns_loc
     FROM ordhead oh,
          ordhead_rev ohr,
          sups s,
          v_restart_supplier v
    WHERE ohr.order_no (+) = oh.order_no
      AND oh.status = 'A'
      AND oh.edi_sent_ind = 'N'
      AND oh.edi_po_ind = 'Y'
      AND oh.supplier = s.supplier
      AND (s.edi_po_chg = 'Y'
          OR (s.edi_po_chg = 'N'
              AND oh.last_sent_rev_no IS NULL))
      AND ohr.origin_type (+) = 'V'
      AND ohr.rev_no(+) = oh.last_sent_rev_no
      and v.driver_name = :ps_restart_driver_name
      and v.driver_value = oh.supplier
      and v.num_threads = :pi_restart_num_threads
      and v.thread_val = :pi_restart_thread_val
    ORDER BY 2 , 3;
```

Restart/recovery capability will be used in this program to provide restart capability. Restartability is implied because the program updates ordhead.edi_sent_ind as records are written out.

Performance Considerations

N/A

Security Considerations

N/A

Design Assumptions

N/A

Outstanding Design Issues

N/A

Appendix

N/A

New and Changed Upload from Supplier [ediupcat]

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.

Scheduling Constraints

Processing Cycle: Daily, Phase 2

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: File-based processing, multithreading not used

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 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. (Similar to the scenario in RMS9.0 that a style could not have a UPC). 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_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.

- Create a 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.
- Create a 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.

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.

Record Name	Field Name	Field Type	Default Value	Description
	Case Unit cost new	Number (20,4)		This field will contain the new unit cost of the bracket for a case UPC.
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	

Conditions	Expected Results	Programmer Sign-off
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.

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

On-order extract [onordext}

Module affected

On-Order Extract – onordext.pc

Design Overview

This program calculates the value in cost and retail of items that are on order for the department/class/subclass/location level. This program is the first step in the stock ledger download process to RPP. It calculates the on order cost and retail for all approved orders that have not before dates less than or equal to the planning horizon date. Once the program has calculated the costs and retails, they are inserted into the ON_ORDER_TEMP table. This table is used in the second step of the stock ledger download process -- the Stock Ledger Extract program (stlgdnld.pc).

Note: The MAX_BUFFER_NUM #define should be set to a number that will allow safe buffering of component pack items, allocations, and orders at the component sku location level. If it is possible for any of these variables to exceed the value assigned to MAX_BUFFER_NUM, its value must be increased.

Stored Procedures / Shared Modules (Maintainability)

Convert_to_primary – Pro*c library (utils.h / utils.pc).

Program Flow

Function Level Description

main():

The standard Retek main() function. Calls init(), process(), and final().

init():

Initialize restart recovery by calling retek_init(). Opens the input file and reads the planning horizon end date. Gets the primary currency from the system_options table.

process():

There are two process loops, one for transaction level items and one for pack items.

For each record brought back by the item driving cursor:

- Convert the cost from the order currency into the primary currency by calling the convert_to_primary function.
- Convert the retail from the location currency into the primary currency by calling the convert_to_primary function.
- Call the handle_alloc function.

After all the record from the item cursor have been processed. The process flag should be flipped to ‘P’ack. The handle_alloc function is called to clean up the last allocation.

For each record brought back by the pack driving cursor:

- Convert the pack cost from the order currency into the primary currency by calling the convert_to_primary function.
- Convert the pack retail from the location currency into the primary currency by calling the convert_to_primary function.
- Convert the pack component item cost from the supplier currency into the primary currency by calling the convert_to_primary function.
- Convert the pack component item retail from the zone/zone_group currency into the primary currency by calling the convert_to_primary function.
- Call the handle_pack function.

Handle_insert()

This function adds records to arrays that will be inserted into ON_ORDER_TEMP. First it populated a record into the regular array (non-allocation). Then if the current record is associated with an allocation, a record is added to the allocation array.

Add_insert_record()

This function copies records from a driving cursor record into an insert array. It has two modes, the first if for non-allocation records. The cost and retail are set to the converted cost and retail from the driving cursor times the item qty from ordloc. The second mode handles allocation records, it sets the cost and retail to the converted cost and retail from the driving cursor times the alloc qty from alloc_detail.

Handle_alloc()

This function sums up the quantities allocated to on alloc_header for each allocation. Once every record associated with an allocation have been summed, it is possible to determine how much of the qty ordered to the allocation warehouse should be assigned to it. If all of the items ordered to it have been allocated, the warehouse is not assigned any of the cost or retail. If items are not fully allocated, the warehouse is assigned the items that were not allocated. Once the qty to give to the warehouse it determined, the handle_insert is function is called for each allocation record.

Handle_pack()

This function prorates the amount and cost ordered through packs to their component items. Every component item ordered through a pack has their total cost and retail summed up. Once all the component items have been summed, the cost and retail of the pack is prorated. The formula cost is: (the pack's cost / the summed component cost) * the individual component item's cost. The same formula is used for retail.

When summing up retails, the class_vat_ind of the pack and its components are considered. When they are different, the components retail is converted to match the pack's. This logic is contained in handle_comp_vat().

After the records are prorated they are sent to buffer_alloc().

Handle_comp_vat()

This function adds or removes vat from pack component's retail as needed. This conversion is only needed when vat is defined as the class level in the system (system_options.class_level_vat_ind).

If the pack's dept/class does not match the component's dept/class:

If the pack's class_vat_ind (defined at the class table) does not match the component's class_vat_ind:

Add or remove vat from the component's retail based on the pack's class_vat_ind. The conversion is done using library function defined in common.h.

Buffer_alloc()

This function is used to reorder the records sent to it by handle_pack() before they are sent to handle_alloc(). The handle_pack() function requires that the records be ordered by order_no, pack_no, location, alloc location. Handle_alloc() requires the records to be ordered by order_no, pack_no, location, component item. This function build groups of order_no/pack_no/location and reorders them by component item. sort_for_allocation(), get_max_indx(), swap_driv_array(), and copy_info() are utility functions that help sort the driving cursor records.

Once the buffered records have been ordered, they are sent to the handle_alloc() function.

Copy_fetch()

Utility function used by handle_alloc and handle_pack. It totals up the allocated qty for the handle_alloc funciton. It totals up the components cost and retail (times the pack item qty) for handle_pack.

Insert_on_order_temp()

This function performs an array insert into the on_order_temp table.

Size_driv_cur_struct()

Allocates memory for the structure used to fetch the driving cursors and used when summing up allocations and pack component items.

Size_insert_struct()

Allocates memory used to perform bulk inserts into the on_order_temp table.

load_dept_class()

Load an array with every dept/class and their class_vat_ind. We want to make sure that all components include or exclude vat depending on whether or not the pack includes or excludes vat. Vat should be consistent across all the retail values when prorating the pack's retail to the components.

To avoid extra-hits database hits, the entire class table is cached with necessary vat information. That way when we need to get a particular class_vat_ind, we can search this array rather than hit the database in our driving cursor loop.

size_dept_class()

Size an array to hold every dept/class/class_vat_ind.

get_dept_class_ind()

Given a dept/class, this function will perform a binary search on the dept/class array and return the class_vat_ind for the passed in dept/class.

get_vat_rate()

Wrapper for call to VAT_SQL.GET_VAT_RATE.

Input Specifications

Command Line Parameters:

Onordext will calculate the on order cost and retail for a given planning horizon. The planning horizon date will be a command line parameter contained in an input file. The file will also contain information used by stlgdnld.pc.

The file contains 1 line

Field Position: 1 – weekly or historic indicator

2-9 – planning horizon start date

10-17 – planning horizon end date

Onordext userid/passwd input-file

Driving Cursor:

There are two driving cursors in this program. The first deals with items, the second deals with packs.

```
      /*
         The first driving cursor handles on-order amounts for items.
         The first part deals with non-allocation ordeers or allocation
   orders
         that are not pre-marked
      */
         EXEC SQL DECLARE c_driver CURSOR FOR
        SELECT oh.order_no,
               -999 alloc_no,
               ol.item,
               ol.qty_ordered - nvl(ol.qty_received,0),
               ol.unit_cost,
               ol.unit_retail,
               ol.location,
               ol.loc_type,
               -999 alloc_loc,
               'N' alloc_loc_type,
               0 alloc_qty,
               im.dept,
                 im.class,
                 im.subclass,
                 oh.exchange_rate,
                 oh.currency_code,
                 oh.otb_eow_date,
                 1 comp_item_qty
            FROM ordhead oh,
                 ordloc ol,
                 item_master im
           WHERE oh.status   = 'A'
             AND ol.qty_ordered > nvl(ol.qty_received,0)
             AND oh.not_before_date <= TO_DATE(:ps_on_order_date,
'YYYYMMDD')
             AND oh.order_no = ol.order_no
             AND ol.item      = im.item
             AND im.pack_ind = 'N'
             AND (oh.pre_mark_ind = 'N' OR
                  (oh.pre_mark_ind = 'Y'
                   and not exists (select alloc_no
                                     from alloc_header ah
                                    where ah.order_no = oh.order_no
                                      and ol.item      = ah.item
                                      and ol.location = ah.wh)))
             AND oh.order_no > NVL(:ps_restart_order, -999)
             AND MOD(oh.order_no, TO_NUMBER(:ps_restart_num_threads)) + 1 =
TO_NUMBER(:ps_restart_thread_val)
   UNION ALL
          SELECT oh.order_no,
                 NVL(ah.alloc_no,-999),
                 ol.item,
                 ol.qty_ordered - nvl(ol.qty_received,0),
                 ol.unit_cost,
                 ol.unit_retail,
                 ol.location,
                 ol.loc_type,
                 ad.to_loc,
                 ad.to_loc_type,
                 nvl(ad.qty_allocated, 0) - nvl(ad.qty_transferred,0),
                 im.dept,
                 im.class,
                 im.subclass,
                 oh.exchange_rate,
                 oh.currency_code,
                 oh.otb_eow_date,
                  1 comp_item_qty
             FROM ordhead oh,
                  ordloc ol,
                  item_master im,
                  alloc_header ah,
                  alloc_detail ad
            WHERE oh.status   = 'A'
              AND ol.qty_ordered > nvl(ol.qty_received,0)
              AND oh.pre_mark_ind = 'Y'
              AND oh.not_before_date <= TO_DATE(:ps_on_order_date,
'YYYYMMDD')
              AND oh.order_no = ol.order_no
              AND ol.item      = im.item
              AND im.pack_ind = 'N'
              AND ol.order_no = ah.order_no
              AND ol.location = ah.wh
              AND ol.item      = ah.item
              AND ah.alloc_no = ad.alloc_no
              AND oh.order_no > NVL(:ps_restart_order, -999)
              AND MOD(oh.order_no, TO_NUMBER(:ps_restart_num_threads)) + 1 =
TO_NUMBER(:ps_restart_thread_val)
            ORDER BY 1,2,7,9;


   /*
     The second driving cursor handles on-order amounts for packs.
     - The first part deals with packs being ordered directly to a
store.
     - The second part deals with packs being directly ordered to a wh
or
         allocated from a wh with the pre-mark indicator set to 'N'.
     - The third part deals with packs being allocated from a wh with
the
         pre-mark indicatory set to "Y'.
   */
   EXEC SQL DECLARE c_pack_driver CURSOR FOR
      SELECT oh.order_no,
             -999 alloc_no,
             ol.item pack_no,
             ol.qty_ordered - nvl(ol.qty_received,0) pack_qty,
             ol.unit_cost pack_cost,
             ol.unit_retail pack_retail,
             vpq.item,
             vpq.qty item_qty,
             iscl.unit_cost comp_cost,
             izp.unit_retail comp_retail,
                ol.location,
                ol.loc_type,
                -999 alloc_loc,
                'N' alloc_loc_type,
                0 alloc_qty,
                im.dept,
                im.class,
                im.subclass,
                iscl.supplier,
                izp.zone_id,
                izp.zone_group_id,
                oh.exchange_rate,
                oh.currency_code,
                oh.otb_eow_date,
                vpq.qty comp_item_qty,
                to_char(imp.dept,'0000')||trim(to_char(imp.class,'0000')),
                to_char(im.dept,'0000')||trim(to_char(im.class,'0000'))
           FROM ordhead oh,
                ordsku os,
                ordloc ol,
                item_master im,
                item_master imp,
                v_packsku_qty vpq,
                item_supp_country_loc iscl,
                price_zone_group_store pzgs,
                item_zone_price izp
          WHERE oh.status           = 'A'
            AND ol.qty_ordered > nvl(ol.qty_received,0)
            AND oh.not_before_date <= TO_DATE(:ps_on_order_date,
'YYYYMMDD')
            AND oh.pre_mark_ind     = 'N'
            AND oh.order_no         = os.order_no
            AND oh.order_no         = ol.order_no
            AND ol.loc_type         = 'S'
            AND os.item               = imp.item
            AND os.item              = ol.item
            AND ol.item              = vpq.pack_no
            AND im.item              = vpq.item
            AND iscl.item            = im.item
            AND oh.supplier         = iscl.supplier
            AND os.origin_country_id = iscl.origin_country_id
            AND pzgs.store          = ol.location
            AND pzgs.zone_id        = izp.zone_id
          AND pzgs.zone_group_id = izp.zone_group_id
          AND izp.item           = vpq.item
          AND oh.order_no > NVL(:ps_restart_order, -999)
          AND MOD(oh.order_no, TO_NUMBER(:ps_restart_num_threads)) + 1 =
TO_NUMBER(:ps_restart_thread_val)
   UNION ALL
      SELECT oh.order_no,
             -999 alloc_no,
             ol.item pack_no,
             ol.qty_ordered - nvl(ol.qty_received,0) pack_qty,
             ol.unit_cost pack_cost,
             ol.unit_retail pack_retail,
             vpq.item,
             vpq.qty item_qty,
             iscl.unit_cost comp_cost,
             izp.unit_retail comp_retail,
             ol.location,
             ol.loc_type,
             -999 alloc_loc,
             'N' alloc_loc_type,
             0 alloc_qty,
             im.dept,
             im.class,
             im.subclass,
             iscl.supplier,
             izp.zone_id,
             izp.zone_group_id,
             oh.exchange_rate,
             oh.currency_code,
             oh.otb_eow_date,
             vpq.qty comp_item_qty,
             to_char(imp.dept,'0000')||trim(to_char(imp.class,'0000')),
             to_char(im.dept,'0000')||trim(to_char(im.class,'0000'))
        FROM ordhead oh,
             ordsku os,
             ordloc ol,
             item_master im,
             item_master imp,
             v_packsku_qty vpq,
             item_supp_country_loc iscl,
             item_zone_price izp
       WHERE oh.status           = 'A'
         AND ol.qty_ordered > nvl(ol.qty_received,0)
         AND oh.not_before_date <= TO_DATE(:ps_on_order_date,
'YYYYMMDD')
         AND oh.order_no        = os.order_no
         AND oh.order_no        = ol.order_no
         AND os.item              = imp.item
         AND os.item             = ol.item
         AND ol.loc_type        = 'W'
         AND ol.item             = vpq.pack_no
         AND im.item             = vpq.item
         AND iscl.item           = im.item
         AND oh.supplier        = iscl.supplier
         AND os.origin_country_id = iscl.origin_country_id
         AND izp.item            = vpq.item
         AND izp.base_retail_ind = 'Y'
         AND (oh.pre_mark_ind = 'N' OR
              (oh.pre_mark_ind = 'Y'
               and not exists (select alloc_no
                                 from alloc_header ah
                                where ah.order_no = oh.order_no
                                  and ol.item      = ah.item
                                  and ol.location = ah.wh)))
         AND oh.order_no > NVL(:ps_restart_order, -999)
         AND MOD(oh.order_no, TO_NUMBER(:ps_restart_num_threads)) + 1 =
TO_NUMBER(:ps_restart_thread_val)
   UNION ALL
      SELECT oh.order_no,
             NVL(ah.alloc_no,-999),
             ol.item pack_no,
             ol.qty_ordered - nvl(ol.qty_received,0) pack_qty,
             ol.unit_cost pack_cost,
             ol.unit_retail pack_retail,
             vpq.item,
             vpq.qty item_qty,
             iscl.unit_cost comp_cost,
             izp.unit_retail comp_retail,
             ol.location,
             ol.loc_type,
             ad.to_loc,
             ad.to_loc_type,
             nvl(ad.qty_allocated, 0) - nvl(ad.qty_transferred,0),
             im.dept,
             im.class,
             im.subclass,
             iscl.supplier,
                    izp.zone_id,
                    izp.zone_group_id,
                    oh.exchange_rate,
                    oh.currency_code,
                    oh.otb_eow_date,
                    vpq.qty comp_item_qty,
                    to_char(imp.dept,'0000')||trim(to_char(imp.class,'0000')),
                    to_char(im.dept,'0000')||trim(to_char(im.class,'0000'))
               FROM ordhead oh,
                    ordsku os,
                    ordloc ol,
                    item_master im,
                    item_master imp,
                    v_packsku_qty vpq,
                    item_supp_country_loc iscl,
                    item_zone_price izp,
                    alloc_header ah,
                    alloc_detail ad
              WHERE oh.status           = 'A'
                AND ol.qty_ordered > nvl(ol.qty_received,0)
                AND oh.pre_mark_ind    = 'Y'
                AND oh.not_before_date <= TO_DATE(:ps_on_order_date,
'YYYYMMDD')
                AND oh.order_no         = os.order_no
                AND oh.order_no         = ol.order_no
                AND os.item              = imp.item
                AND os.item              = ol.item
                AND ol.loc_type         = 'W'
                AND ol.item              = vpq.pack_no
                AND im.item              = vpq.item
                AND iscl.item            = im.item
                AND oh.supplier         = iscl.supplier
                AND os.origin_country_id = iscl.origin_country_id
                AND izp.item            = vpq.item
                AND izp.base_retail_ind = 'Y'
                AND ol.order_no         = ah.order_no
                AND ol.location         = ah.wh
                AND ol.item              = ah.item
                AND ah.alloc_no         = ad.alloc_no
                AND oh.order_no > NVL(:ps_restart_order, -999)
                AND MOD(oh.order_no, TO_NUMBER(:ps_restart_num_threads)) + 1 =
TO_NUMBER(:ps_restart_thread_val)
             ORDER BY 1,2,3,13,15;
```

Output Specifications

ON_ORDER_TEMP will be populated by this program.

ITEM	NOT NULL VARCHAR2(25)
DEPT	NOT NULL NUMBER(4)
CLASS	NOT NULL NUMBER(4)
SUBCLASS	NOT NULL NUMBER(4)
OTB_EOW_DATE	NOT NULL DATE
STORE	NOT NULL NUMBER(10)
WH	NOT NULL NUMBER(10)
ON_ORDER_RETAIL	NOT NULL NUMBER(20,4)
ON_ORDER_COST	NOT NULL NUMBER(20,4)
ON_ORDER_UNITS	NOT NULL NUMBER(20,4)

Scheduling Considerations

This program can be run weekly in Phase 4.

Prepost onordext pre must run before this program.

This program should be run before onorddnld.pc.

Locking Strategy

Restart/Recovery

Logical unit of work (LUW) is a unique order number. This is a non-unique LUW.

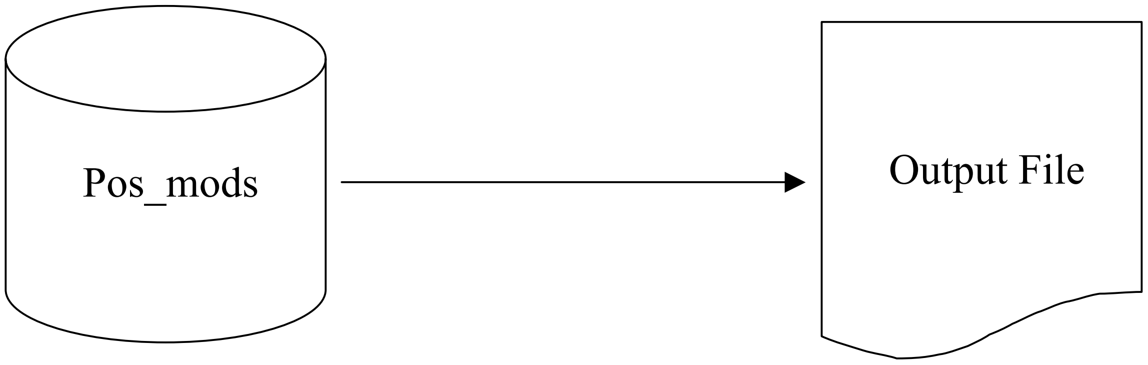
It is also split into two sections item and pack. First all items on orders are processed. When they are done a pack ‘flag’ is turned on and the restart order is reset. Then all the packs on order are processed. So all orders are considered twice, once for items and once for packs.

POS download [posdnld]

Design overview

The posdnld program is used to download pos_mods records created in the RMS to the store POS systems. This program has one output file which contains all records for all stores in a given run. This program uses the Retek standard file format FHEAD, FDETL, FTAIL.

Program Flow



Stored Procedures / Shared Modules (Maintainability)

pos_config_sql.check_item - Updates POS item configuration information that is downloaded to the stores by poscdnld.pc.

Input Specifications

All input comes from the pos_mods table. All columns of this table can be NULL with the exception of tran_type and store. Most columns should default to blank (spaces) with the exception of:

- new_price, new_multi_units, new_multi_units_retail, proportional_tare_pct and fixed_tare_value. These should default to zero (0).
- start_date, start_time and end_time. These should default to period.vdate + 1.

Output Specifications

Output File

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	Number ID(10)	Sequential number Created by program.	ID of current line being created for output file.

Record Name	Field Name	Field Type	Default Value	Description
	File Type Definition	Char(4)	POSD	Identifies file as 'POS Download'
	File Create Date	Char(8)	Create date (vdate).	Current date, formatted to 'YYYYMMDD'.
File Detail	**File Type Record Descriptor**	**Char(5)**	**FDETL**	**Identifies file record type**
	File Line Identifier	Number ID(10)	Sequential number. Created by program.	ID of current line being created for output file.
	Location Number	Number(1 0)	Store	Contains the store location that has been affected by the transaction
	Update Type	Char(1)	Update type. Created by program.	Code used for client specific POS system. 1 - Transaction Types 1 & 2. 2 - Transaction Types 10 thru 18, 31 & 32, 50 thru 57, 59 thru 64. 3 - Transaction Types 21 & 22 4 - Transaction Types 25 & 26 0 - All other Transaction Types. These should never exist.
	Start Date	Char(8)	Start_date or vdate + 1 if NULL.	The effective date for the action determined by the transaction type of the record. Formatted to 'YYYYMMDD'.
	Time	Char(6)	Start_time, End_time or start_date.	This field will be used in conjunction with starting a promotion (Transaction Type = 31). Start time will indicate the time of day that the promotion is scheduled to start. This field will also be used in conjunction with ending a promotion (Transaction Type = 32). Any other Transaction Type will use the time from the start_date column. Formatted to 'HH24MISS'.
	Transaction Type	Number(2)	Tran_type	Indicates the type of transaction to determine what Retek action is being sent down to the stores from

Record Name	Field Name	Field Type	Default Value	Description
				the Retek pos_mods table. Valid values include: 01 - Add new transaction level item 02 - Add new lower than transaction level item 10 - Change Short Description of existing item 11 - Change Price of an existing item 12 - Change Description of an existing item 13 - Change Department/Class/Subclass of an existing item 16 - Put Item on Clearance 17 - Change existing item's Clearance Price 18 - Remove Item from Clearance and Reset 20 - 21 - Delete existing transaction level item 22 - Delete existing lower than transaction level item 25 - Change item's status 26 - Change item's taxable indicator 31 - Promotional item - Start maintenance 32 - Promotional item - End maintenance 50 - Change item's launch date 51 - Change item's quantity key options 52 - Change item's manual price entry options 53 - Change item's deposit code 54 - Change item's food stamp indicator 55 - Change item's WIC indicator 56 - Change item's proportional

Record Name	Field Name	Field Type	Default Value	Description
				tare percent 57 - Change item's fixed tare value 58 - Change item's rewards eligible indicator 59- Change item's electronic marketing clubs 60 - Change item's return policy 61 - Change item's stop sale indicator 62 – Change item’s returnable indicator
	Item Number ID	Char(25)	Item	This field identifies the unique alphanumeric value for the transaction level item. The ID number of a item from the Retek item_master table.
	Item Number Type	Char(6)	Item_number_type	This field identifies the type of the item number ID.
	Format ID	Char(1)	Format_id	This field identifies the type of format used if the item_number_type is ‘VPLU’.
	Prefix	Number(2)	Prefix	This field identifies the prefix used if the item_number_type is ‘VPLU’. In case of single digit prefix, the field will be right-justified with blank padding.
	Reference Item	Char(25)	Ref_item	This field identifies the unique alphanumeric value for an item one level below the transaction level item.
	Reference Item Number Type	Char(6)	Ref_Item_number_type	This field identifies the type of the ref item number ID.
	Reference Item Format ID	Char(1)	Ref_Format_id	This field identifies the type of format used if the ref item_number_type is ‘VPLU’.
	Reference Item Prefix	Number(2)	Ref_Prefix	This field identifies the prefix used if the ref item_number_type is ‘VPLU’. In case of single digit prefix, the field will be right-justified with blank padding.

Record Name	Field Name	Field Type	Default Value	Description
	Item Short Description	Char(20)	Item_short_desc	Contains the short description associated with the item.
	Item Long Description	Char(100)	Item_long_desc	Contains the long description associated with the item.
	Department ID	Number(4)	Dept	Contains the item's associated department.
	Class ID	Number(4)	Class	Contains the item's associated class.
	Subclass ID	Number(4)	Subclass	Contains the item's associated subclass.
	New Price	Number(20)	New_price	Contains the new effective price in the selling unit of measure for an item when the transaction type identifies a change in price. Otherwise, the current retail price is used to populate this field. This field is stored in the local currency.
	New Selling UOM	Char(4)	New_selling_UOM	Contains the new selling unit of measure for an item's single-unit retail.
	New Multi Units	Number(12)	New_multi_units	Contains the new number of units sold together for multi-unit pricing. This field is only filled when a multi-unit price change is being made.
	New Multi Units Retail	Number(20)	New_multi_units_retail	Contains the new price in the selling unit of measure for units sold together for multi-unit pricing. This field is only filled when a multi-unit price change is being made. This field is stored in the local currency.
	New Multi Selling UOM	Char(4)	New_multi_selling_UOM	Contains the new selling unit of measure for an item's multi-unit retail.

Record Name	Field Name	Field Type	Default Value	Description

	Status	Char(1)	Status	Populates if tran_type for the item is 1(new item added) or 25 (change item status) or 26 (change taxable indicator).
				Contains the current status of the item at the store.
				Valid values are:
				A = Active
				I = Inactive
				D = Delete
				C = Discontinued

	Taxable Indicator	Char(1)	Taxable_ind	Populates if tran_type for the item is 1 (new item added) or 25 (change item status) or 26 (change taxable indicator). Indicates whether the item is taxable at the store. Valid values are 'Y' or 'N'.
	Promotion Number	Number(10)	Promotion	This field contains the number of the promotion for which the discount originated. This field, along with the Mix Match Number or Threshold Number is used to isolate a list of items that tie together with discount information.
	Mix Match Number	Number(10)	Mix_match_no	This field contains the number of the mix and match in a promotion for which the discount originated. This field, along with the promotion, is used to isolate a list of items which tie together with the mix and match discount information.

Record Name	Field Name	Field Type	Default Value	Description
	Mix Match Type	Char(1)	Mix_match_type	This field identifies which types of mix and match record this item belongs to. The item can either be a buy (exists on PROM_MIX_MATCH_BUY) or a get (exists on PROM_MIX_MATCH_GET) item. This field is only populated when the MIX_MATCH_NO is populated. Valid values are: B - Buy G - Get
	Threshold Number	Number(10)	Threshold_no	This field contains the number of the threshold in a promotion for which the discount originated. This field, along with the promotion, is used to isolate a list of items that tie together with discount information.
	Launch Date	Char(8)	Launch_date	Date that the item should first be sold at this location, formatted to 'YYYYMMDD'.
	Quantity Key Options	Char(6)	Qty_key_options	Determines whether the price can/should be entered manually on a POS for this item at the location. Valid values are in the code_type 'RPO'. Current values include 'R - required', 'P - Prohibited.
	Manual Price Entry	Char(6)	Manual_price_entry	Determines whether the price can/should be entered manually on a POS for this item at the location. Valid values are in the code_type 'RPO'. Current values include 'R - required', 'P - Prohibited', and 'O - Optional'.

Record Name	Field Name	Field Type	Default Value	Description
	Deposit Code	Char(6)	Deposit_code	Indicates whether a deposit is associated with this item at the location. Valid values are in the code_type 'DEPO'. Additional values may be added or removed as needed. Deposits are not subtracted from the retail of an item uploaded to RMS, etc. This kind of processing is the responsibility of the client and should occur before sales are sent to any Retek application.
	Food Stamp Indicator	Char(1)	Food_stamp_ind	Indicates whether the item is approved for food stamps at the location.
	WIC Indicator	Char(1)	Wic_ind	Indicates whether the item is approved for WIC at the location.
	Proportional Tare Percent	Number(12)	Proportional_tare_pct	Holds the value associated of the packaging in items sold by weight at the location. The proportional tare is the proportion of the total weight of a unit of an item that is packaging (i.e. if the tare item is bulk candy, this is the proportional of the total weight of one piece of candy that is the candy wrapper). The only processing RMS does involving the proportional tare percent is downloading it to the POS.

Record Name	Field Name	Field Type	Default Value	Description
	Fixed Tare Value	Number(12)	Fixed_tare_value	Holds the value associated of the packaging in items sold by weight at the location. Fixed tare is the tare of the packaging used to (i.e. if the tare item is bulk candy, this is weight of the bag and twist tie). The only processing RMS does involving the fixed tare value is downloading it to the POS. Fixed tare is not subtracted from items sold by weight when sales are uploaded to RMS, etc. This kind of processing is the responsibility of the client and should occur before sales are sent to any Retek application.
	Fixed Tare UOM	Char(4)	Fixed_tare_uom	Holds the unit of measure value associated with the tare value. The only processing RMS does involving the proportional tare value and UOM is downloading it to the POS. This kind of processing is the responsibility of the client and should occur before sales are sent to any Retek application.
	Reward Eligible Indicator	Char(1)	Reward_eligible_ind	Holds whether the item is legally valid for various types of bonus point/award programs at the location.
	Elective Marketing Clubs	Char(6)	Elect_mtk_clubs	Holds the code that represents the marketing clubs to which the item belongs at the location. Valid values can belong to the code_type 'MTKC'. Additional values can be added or removed from the code type as needed
	Return Policy	Char(6)	Return_pocily	Holds the return policy for the item at the location. Valid values for this field belong to the code_type 'RETP'.
	Stop Sale Indicator	Char(1)	Stop_sale_ind	Indicates that sale of the item should be stopped immediately at the location (i.e. in case of recall etc).

Record Name	Field Name	Field Type	Default Value	Description
	Returnable Indicator	Char(1)	Returnable_ind	Indicates that the item is returnable at the location when equal to 'Y'es. Indicates that the item is not returnable at the location when equal to 'N'o.
	Refundable Indicator	Char(1)	Refundable_ind	Indicates that the item is refundable at the location when equal to 'Y'es. Indicates that the item is not refundable at the location when equal to 'N'o.
	Back Order Indicator	Char(1)	Back_order_ind	Indicates that the item is back orderable at the location when equal to 'Y'. Indicates that the item is not back orderable when equal to 'N'o.
	Vat Code	Char(6)		Indicates the VAT code used with this item.
	Vat Rate	Number(20,10)		Indicates the VAT rate associated with this item and VAT code.
	Class Vat Indicator	Char(1)		Indicates whether or not the class VAT indicator is on or off for the class that this item exists in.
File Trailer	**File Type Record Descriptor**	**Char(5)**		**Identifies file record type**
	File Line Identifier	Number ID(10)	Sequential number. Created by program.	ID of current line being created for output file.
	File Record Counter	Number ID(10)	Number of FDETL records. Created by program.	Number of records/transactions processed in current file (only records between head & tail)

Function Level Description

init - This function initializes restart/recovery for this program. It also retrieves system variables (period.vdate and vdate + 1), opens the output file and write the FHEAD record.

process - This function drives the processing of the program. It calls size_arrays() function to size the arrays used in this program and also, when done, it calls free_arrays() to release any memory it has been allocated. The driving cursor is opened and fetched here, which retrieves all the records from pos_mods where the pos_mods.store value is greater than zero.

If the Transaction Type is 31, then the time field returned by the cursor should be the start time, else if the Transaction Type is 32, then the time field should be the end time. If the Transaction Type is something else or if either the start time or end time is NULL, blanks should be used.

Once the records are fetched, if the Transaction Type of the record fetched is 1 or 21 then pos_config_check() is called. The write_rec() function is called to perform processing on all records fetched. Restart/Recovery and committing of records is also performed here.

final - This function will finish restart/recovery logic, write the FTAIL record and close the output.

size_arrays - This function initializes the size of the array used for the driving cursor fetch the size of the restart max counter on restart_control.

free_arrays - This function frees the array allocated in size_arrays.

write_rec - This function will prepare records for insert into the output file and write them as FDETL records. The Transaction Type will determine the Update Type. If the Transaction Type is 1, 25 or 26 then the status and taxable_ind columns must be outputted, otherwise these should remain blank.

pos_config_check - This function will call the package pos_config_sql.check_item(). If the Transaction type is 1, then a status of 'A' will be passed in. If the Transaction Type is 21, then a status of 'D' will be passed in. This function body should be commented out for A&P Phase 1a.

Scheduling Considerations

Processing Cycle: PHASE 4 (daily)

Scheduling Diagram: This program is run towards the end of the batch run when all pos_mods records have been created for the transaction day.

Pre-Processing: N/A

Post-Processing: prepost.pc - posdnld_post() – records in POS_MODS are truncated.

Threading Scheme: v_restart_store

Locking Strategy

None.

Restart/Recovery

Restart/recovery for this program is set up at the store/item or item level. Threading is done by store using the v_restart_store view to thread properly.

Performance Considerations

Both table and file restart/recovery must be used.

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).

Security Considerations

Price changes for all stores are stored in a Unix file with the processes default permissions (umask). Care should be exercised so that this file cannot be tampered with.

Design Assumptions

Data columns required by a particular Transaction Type are filled in and correct.

Outstanding Design Issues

The columns pos_config_item.item and pos_merch_criteria.sku are of type number and have a length of 8. These columns are updated and referenced by the pos_config_sql.check_item() package function. These tables are then used by poscdnld.pc.

Appendix

None.

POS Upload [posupld]

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.

Scheduling Constraints

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.

Program Flow

N/A

Shared Modules

validate_all_numeric: intrface library function.

validate_all_numeric_signed: intrface library function.

valid_date: intrface library function.

ORDER_ATTRIB_SQL.DELIVERY_MONTH: called from consignment_data(), returns order delivery month into the :invoices variable.

VAT_SQL.GET_VAT_RATE: called from pack_check(), 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() 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.

Posupld and VAT:

There are three different data sources in POSUPLD.

1 The input file

2 RMS stock ledger tables (tran_data in this context)

3 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.

1 system_options.vat_ind (assume Y for this document)

2 system_options.class_level_vat_ind

3 system_options.stkldgr_vat_incl_retl_ind

4 class.class_vat_ind

5 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

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	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
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

Declarations:

declare input structures: file header (only date and type) & detail (all fields)

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.

Retrieve all valid promotion types

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 discrepencies 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.

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, the transaction will be written 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 it's price_hist record. Since we do not create price_hist records of type 9 (promotional retail change) when the system_options.multi_prom_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 system_options.multi_prom_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.

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 system_options.multi_prom_ind = 'N' and the item is on promotion, or if the system_options.multi_prom_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, system_options.multi_prom_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 system_options.multi_prom_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.

process_sales_and_returns()

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 will be called to write a tran_data record with a tran_type 1 (always written), a tran_type 2 (if the system_options.stkldgr_vat_incl_retl_ind = Y), 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.

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.

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.

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 tranasaction 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.

Check_item_lock

This function will lock this item/location's record in the RMS item_loc table. Returns a lock error if lock failed due to contention, otherwise returns 0 if no errors occurred, or fatal if other errors occurred.

Check_pack_lock

This function will call check_item_lock for every component item of the current pack item.

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 a promotion number is passed in, validate it by checking the promhead table and set the promotional indicator 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.

New_item_loc

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

item_store_cursors

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

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.

New_pack_loc

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

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	
packitem A	$60				
item 1	$15	2	$30	($30/$90) * $60	.3333
item 2	$10	6	$60	($60/$90) * $60	.6667

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.

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.

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_exist and daily_sales_insert_update.

Remove_stklgdr_vat()

This fuction will remove vat from 3 fields after the dailiy_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()

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.

Daily_sales_exist()

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

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.

write_off_retail_markdown()

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_markdown()

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.

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.

vat_convert()

This function will either add or remove vat from a retail value.

process_items()

Update the stock on hand on the item_loc_soh table for Sales and Returns unless the item is on consignment. Also, update the item_loc_hist table for Sale transactions. Do not update for returns.

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. Do not update for returns.

write_tran_data()

Writes a record to the tran_data insert array.

Write_edi_daily_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.

write_vat_err_message()

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

vat_history_data()

Writes a record to the vat_history table.

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

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.

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'.

fill_packitem_array()

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

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.

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.

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 functionby 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
	Currency code	Char(3)		Currency of the given location. Not required but populated by Retek sales audit

Record Name	Field Name	Field Type	Default Value	Description
	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)**		**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
	Pack Indicator	Char(1)	Item's pack indicator	Pack indicator of item sold or returned. Not required but populated by Retek sales audit

Record Name	Field Name	Field Type	Default Value	Description
	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.
	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.

Record Name	Field Name	Field Type	Default Value	Description
	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
Transaction Detail	**File Type Record Descriptor**	**Char(5)**	**TDETL**	**Identifies transaction record type**
	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.

Record Name	Field Name	Field Type	Default Value	Description
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
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.

Complex Deals Management [precostcalc]

Design Overview

This batch module is responsible for data maintenance tasks that are necessary before running costcalc.

Unprocessed records in RECLASS_COST_CHG_QUEUE drive this program. The driving cursor is a set of seven cursors whose goal is to make sure the DEAL_SKU_TEMP table is inserted for three scenarios:

a If an unprocessed record exists on RECLASS_COST_CHG_QUEUE, create record(s) on DEAL_SKU_TEMP for the RECLASS_COST_CHG_QUEUE event. For reclassification events no location is given, therefore these events are blown out to all item-location locations before inserting to DEAL_SKU_TEMP.

b If an unprocessed record exists on RECLASS_COST_CHG_QUEUE, create record(s) on DEAL_SKU_TEMP for any events on the FUTURE_COST table that the RECLASS_COST_CHG_QUEUE event affects (make the match using item, supplier, origin country id, location, and start date). For reclassification events (not cost change) no location is given and therefore these events are blown out to all item-location locations before seeking FUTURE_COST matches.

c If a record exists on DEAL_SKU_TEMP that affects one or more FUTURE_COST records insert for those records on FUTURE_COST into DEAL_SKU_TEMP to make sure they are recalculated.

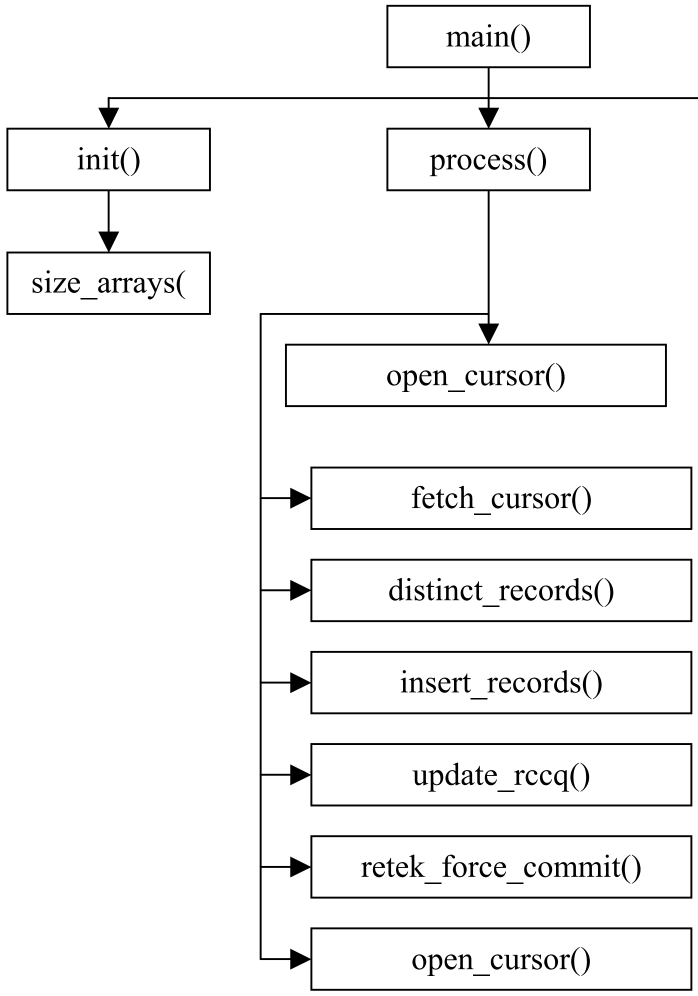
The program will also update RECLASS_COST_CHG_QUEUE events that it processed so that the record’s process flag reflects the fact that it has been processed. To avoid primary key violations in DEAL_SKU_TEMP, check if an item, location, supplier, origin country id, and start date combination exists on DEAL_SKU_TEMP in the driving cursors. When the program is done, it should delete any records from RECLASS_COST_CHG_QUEUE that did not get processed. (This happens when somebody did a cost change or reclassified a component item of a case UPC. This record will not be picked up by the driving cursor and should not remain on RECLASS_COST_CHG_QUEUE.) Also, all records that are of type G or N should also be deleted since once they have been inserted into DEAL_SKU_TEMP, There is no reason for them to remain in RECLASS_COST_CHG_QUEUE. This should improve the performance of the program by keeping the size of RECLASS_COST_CHG_QUEUE as small as possible.

The LUW of this module is a single record from any one of the driving cursors, which all pick up item/supplier/origin country/location/active date combinations from RECLASS_COST_CHG_QUEUE, and FUTURE_COST.

Stored Procedures / Shared Modules (Maintainability)

N/A

Program Flow



Function Level Description

Declare a fetch array struct to hold array-fetched records from driving cursors.

Declare driving cursors The first eight cursors pick up unprocessed events from RECLASS_COST_CHG_QUEUE that do not exist in DEAL_SKU_TEMP. There is eight of them because each handles a different type of record depending on whether location is given or not, and whether division is given or not. So, the combinations that are possible and therefore need to be fetched are:

Location given?	Division given?
N	N
N	Y
Y	N

Remember that the driving cursors for every one of the four above described combinations will be again split into two cursors depending on whether the location is a warehouse or a store. The reason so many driving cursors exist is because a single cursor that joined to all the tables at once was so slow that using such a cursor here was not an option. One more cursor is required: the ninth cursor will fetch records from the FUTURE_COST table that have the same item/supplier/origin country/location combination as a DEAL_SKU_TEMP record with an active date that is greater than the DEAL_SKU_TEMP record's start date. This cursor is necessary to ensure that when an event is inserted into DEAL_SKU_TEMP, later events for this event in the FUTURE_COST table may be affected by the event in DEAL_SKU_TEMP and should be re-inserted into DEAL_SKU_TEMP so costcalc can recalculate and reinsert them into FUTURE_COST.

The cursors should not pick up any primary cost pack component items since those are never inserted into DEAL_SKU_TEMP. (See Design Assumptions below.) Also this cursor should never pick up items that are not approved, transaction level, or items that are buyer packs. Deals batch programs do not process those.

Main():
Standard Retek main function. Validates input parameters, calls init, process and final. Logs appropriate message.

Init():
Standard Retek init function. Calls retek_init() and size_arrays(). If this is a new start, set the restart cursor variable to 1.

Process():
This is the main function that does all the work. It will array fetch each of the 9 cursors in the order of their definition above, one at a time until cursor is empty. For each cursor, fetched records will be array inserted into DEAL_SKU_TEMP, RECLASS_COST_CHG_QUEUE will be updated after every second cursor fetch to set the process_flag of covered records to 'Y' and commit logic will be executed. After the last cursor has been fetched empty and all records were processed, delete from RECLASS_COST_CHG_QUEUE all records that belong to this thread and have not been picked up by any of the cursors (their process_flag is still 'N'). Also delete any records whose rec_type is 'G' or 'N'.

- Call open_cursor to open the first cursor. (Cursors are opened and fetched in the order that they were declared in, which should be the same order they were described in above.)
- In a while loop (while the no records found indicator is not set):
    - Array fetch records from current driving cursor by calling fetch_cursor().
    - Call distinct_records() to copy distinct (item/supplier/origin country/location/start date) records from the fetch array (which has RECLASS_COST_CHG_QUEUE's level of distinctness: item/supplier/origin country/location/start date/record type) to the insert array.
    - Call insert_records() to array insert records from the insert array.

  - If we fetched empty a cursor with an even number call update_rccq() to update RECLASS_COST_CHG_QUEUE's process_flags to 'Y' for the records covered by the last two cursors.
  - Call retek_force_commit() with the item, supplier and origin country, location, start date, and cursor number of the last record as the argument.
  - If the no records found indicator was set by fetch_cursor(), increment the cursor counter. If the cursor counter is under 10, call open_cursor() and reset the no records found indicator to 0 along with the total fetched records counter, which should be zeroed since we are about to start fetching from a new cursor. If the cursor number is 10 or above, we are done. Do not call open_cursor() and leave the no records found indicator set. This will drop the process out of the while loop.

- Delete from RECLASS_COST_CHG_QUEUE all records for this thread where the process_flag is 'N'. (This may happen if the component item of a case UPC was inserted into RECLASS_COST_CHG_QUEUE. The driving cursor will not pick these records up.) Also delete for records whose rec_type is 'G' or 'N' regardless of their process_flag. We do not need to keep those events around in RECLASS_COST_CHG_QUEUE.

Insert_records():
This function performs an array insert into DEAL_SKU_TEMP. Its argument is the current array size (number of records fetched into array).

Distinct_records():
This function copies records from the fetch array (which has RECLASS_COST_CHG_QUEUE's level of distinctness: item/supplier/origin country/location/start date/record type) to the insert array (which has DEAL_SKU_TEMP's level of distinctness: item/supplier/origin country/location/start date) so that primary key/index violation errors are avoided on subsequent inserts into DEAL_SKU_TEMP. This function will need local static strings to store any previous call's last copied DEAL_SKU_TEMP primary key so that it can be compared to the next call's first fetch array element's DEAL_SKU_TEMP primary key fields (item/supplier/origin country/location/start date). This is necessary if an array fetch cut the return of two identical item/supplier/origin country/location/start date records into two array blocks because of the max commit counter. We want to keep track of the fact that the item/supplier/origin country/location/start date record in question was already in the previous block. If this block still has that item/supplier/origin country/location/start date in the first field(s), don't copy it into the insert array again.

Open_cursor():
This function consists of a case-switch statement that depending on the argument of the function (an integer representing the number of the cursor that needs to be opened) will open the appropriate cursor.

Fetch_cursor():
This function consists of a case-switch statement that depending on the argument of the function (an integer representing the number of the cursor that needs to be fetched) will fetch the appropriate cursor and returns the number of records fetched, along with an indicator if the no data found indicator has been set by the fetch or not.

Update_rccq():
This function will be called for every other cursor that was fetched empty and it will update RECLASS_COST_CHG_QUEUE records' process_flag to 'Y' covered by the last two cursors. This strange method of updating those records was necessary for restart/recovery reasons. Since two cursors are used to fetch for a single RECLASS_COST_CHG_QUEUE record (i.e.: no location or division given), one fetching for warehouse locations the other for store locations, the process flag of such a record can not be updated to Y until both cursors were fetched for this record. Otherwise the stores would not get picked up if we update the process flag right after warehouses were fetched. The update statements cover all records in RECLASS_COST_CHG_QUEUE for which the most recent two cursors were fetched for, not only the ones for which the cursors actually returned records. A cursor may not return records, even though the RECLASS_COST_CHG_QUEUE records were completely valid, simply because the returned values already exist in DEAL_SKU_TEMP and the cursors have a not exists clause.

Size_arrays():
Sizes the fetch array to the commit size.

Free_arrays():
Frees fetch array.

Final():
Standard Retek final function. Calls free_arrays() and retek_close().

Input Specifications

'Table-To-Table'

Select data from:

Table Name	Column Name	Column Type	Transformation
RECLASS_COST_CHG_QUEUE	ITEM	VARCHAR2(25)	NONE
RECLASS_COST_CHG_QUEUE	SUPPLIER	NUMBER(10)	NONE
RECLASS_COST_CHG_QUEUE	ORIGIN_COUNTRY_ID	VARCHAR2(3)	NONE
RECLASS_COST_CHG_QUEUE	START_DATE	DATE	NONE
RECLASS_COST_CHG_QUEUE	LOCATION	NUMBER(10)	Only if REC_TYPE is not R
RECLASS_COST_CHG_QUEUE	DIVISION	NUMBER(4)	Only if REC_TYPE is R

Table Name	Column Name	Column Type	Transformation
RECLASS_COST_CHG_QUEUE	GROUP	NUMBER(4)	Only if REC_TYPE is R
RECLASS_COST_CHG_QUEUE	DEPT	NUMBER(4)	Only if REC_TYPE is R
RECLASS_COST_CHG_QUEUE	CLASS	NUMBER(4)	Only if REC_TYPE is R
RECLASS_COST_CHG_QUEUE	SUBCLASS	NUMBER(4)	Only if REC_TYPE is R
RECLASS_COST_CHG_QUEUE	REC_TYPE	VARCHAR2(1)	NONE
GROUPS	DIVISION	NUMBER(4)	Only if REC_TYPE is not R
DEPS	GROUP_NO	NUMBER(4)	Only if REC_TYPE is not R
ITEM_MASTER	DEPT	NUMBER(4)	Only if REC_TYPE is not R
ITEM_MASTER	CLASS	NUMBER(4)	Only if REC_TYPE is not R
ITEM_MASTER	SUBCLASS	NUMBER(4)	Only if REC_TYPE is not R
ITEM_MASTER	ITEM_PARENT	VARCHAR2(25)	NONE
ITEM_MASTER	ITEM_GRANDPARENT	VARCHAR2(25)	NONE
ITEM_MASTER	DIFF_1	VARCHAR2(10)	NONE
ITEM_MASTER	DIFF_2	VARCHAR2(10)	NONE
ITEM_MASTER	DIFF_3	VARCHAR2(10)	NONE
ITEM_MASTER	DIFF_4	VARCHAR2(10)	NONE
ITEM_LOC	LOC	NUMBER(10)	Only if REC_TYPE is R
ITEM_LOC	LOC_TYPE	VARCHAR2(1)	NONE

Table Name	Column Name	Column Type	Transformation
AREA	CHAIN	NUMBER(4)	Only if LOC_TYPE is S
REGION	AREA	NUMBER(4)	Only if LOC_TYPE is S
DISTRICT	REGION	NUMBER(4)	Only if LOC_TYPE is S
STORE	DISTRICT	NUMBER(4)	Only if LOC_TYPE is S
FUTURE_COST	ACTIVE_DATE	DATE	NONE
DEAL_SKU_TEMP	ITEM	VARCHAR2(25)	NONE
DEAL_SKU_TEMP	SUPPLIER	NUMBER(10)	NONE
DEAL_SKU_TEMP	ORIGIN_COUNTRY_ID	VARCHAR2(3)	NONE
DEAL_SKU_TEMP	DIVISION	NUMBER(4)	NONE
DEAL_SKU_TEMP	GROUP_NO	NUMBER(4)	NONE
DEAL_SKU_TEMP	DEPT	NUMBER(4)	NONE
DEAL_SKU_TEMP	CLASS	NUMBER(4)	NONE
DEAL_SKU_TEMP	SUBCLASS	NUMBER(4)	NONE
DEAL_SKU_TEMP	ITEM_PARENT	VARCHAR2(25)	NONE
DEAL_SKU_TEMP	ITEM_GRANDPARENT	VARCHAR2(25)	NONE
DEAL_SKU_TEMP	DIFF_1	VARCHAR2(10)	NONE
DEAL_SKU_TEMP	DIFF_2	VARCHAR2(10)	NONE
DEAL_SKU_TEMP	DIFF_3	VARCHAR2(10)	NONE
DEAL_SKU_TEMP	DIFF_4	VARCHAR2(10)	NONE
DEAL_SKU_TEMP	CHAIN	NUMBER(4)	NONE
DEAL_SKU_TEMP	AREA	NUMBER(4)	NONE
DEAL_SKU_TEMP	REGION	NUMBER(4)	NONE
DEAL_SKU_TEMP	DISTRICT	NUMBER(4)	NONE
DEAL_SKU_TEMP	LOCATION	NUMBER(10)	NONE
DEAL_SKU_TEMP	LOC_TYPE	VARCHAR2(1)	NONE

Output Specifications

'Table-To-Table'

Delete from: RECLASS_COST_CHG_QUEUE

Update data on:

Table Name	Column Name	Column Type	Transformation
RECLASS_COST_CHG_QUEUE	PROCESS_FLAG	VARCHAR2(1)	Set to Y.

Insert into:

Table Name	Column Name	Column Type	Transformation
DEAL_SKU_TEMP	ITEM	VARCHAR2(25)	N/A
DEAL_SKU_TEMP	SUPPLIER	NUMBER(10)	N/A
DEAL_SKU_TEMP	ORIGIN_COUNTRY_ID	VARCHAR2(3)	N/A
DEAL_SKU_TEMP	START_DATE	DATE	N/A
DEAL_SKU_TEMP	DIVISION	NUMBER(4)	N/A
DEAL_SKU_TEMP	GROUP_NO	NUMBER(4)	N/A
DEAL_SKU_TEMP	DEPT	NUMBER(4)	N/A
DEAL_SKU_TEMP	CLASS	NUMBER(4)	N/A
DEAL_SKU_TEMP	SUBCLASS	NUMBER(4)	N/A
DEAL_SKU_TEMP	ITEM_PARENT	VARCHAR2(25)	N/A
DEAL_SKU_TEMP	ITEM_GRANDPARENT	VARCHAR2(25)	N/A
DEAL_SKU_TEMP	DIFF_1	VARCHAR2(10)	N/A
DEAL_SKU_TEMP	DIFF_2	VARCHAR2(10)	N/A
DEAL_SKU_TEMP	DIFF_3	VARCHAR2(10)	N/A
DEAL_SKU_TEMP	DIFF_4	VARCHAR2(10)	N/A
DEAL_SKU_TEMP	CHAIN	NUMBER(4)	N/A
DEAL_SKU_TEMP	AREA	NUMBER(4)	N/A
DEAL_SKU_TEMP	REGION	NUMBER(4)	N/A
DEAL_SKU_TEMP	DISTRICT	NUMBER(4)	N/A
DEAL_SKU_TEMP	LOCATION	NUMBER(10)	N/A
DEAL_SKU_TEMP	LOC_TYPE	VARCHAR2(1)	N/A

Scheduling Considerations

This module must be run after ditinsrt and before costcalc in the deals batch cycle.

This module is multi-threaded by supplier. See volume-testing documentation for optimum thread value. (I suggest 15-30 threads.)

Locking Strategy

N/A

Restart/Recovery

This program has restart recovery based on item/supplier/origin country/location/start date/cursor number and is multi-threaded by supplier.

Performance Considerations

The driving cursors should be small enough to be executed fast. If the DEAL_SKU_TEMP table holds too many records as the program runs and the cursors' NOT EXISTS statements are slowing things down because of the size of DEAL_SKU_TEMP (which may very well happen), the only remaining performance enhancement that could go into the program is to simply fetch records from the cursors without checking for duplicates at the time of fetch and check for duplicates at time of insert or handle primary key violations in the insert as a non-fatal error. Or fetch all the records from all the cursors and discretize the fetched records in batch based on DEAL_SKU_TEMP's primary key. These "solutions" may not be a performance enhancement; the second suggestion simply takes away the load of discretizing [This isn't English… Do you know what this word is actually supposed to be? – BES] records from the database and keeps it in the batch. Therefore other programs will not suffer a performance loss from the database being slow while precostcalc runs.

Security Considerations

N/A

Unit Test Considerations

When program is tested, tester will probably need to run costcalc and prepost for complete results. The tables DEAL_SKU_TEMP and RECLASS_COST_CHG_QUEUE should not be modified in any way during and between the two program runs outside the programs themselves.

See the program's UTP for further instructions.

Design Assumptions

Background: Costcalc is driven by the DEAL_SKU_TEMP table, which holds item/supplier/origin country/location/active date records that then need to be moved to the FUTURE_COST table with their costs at the date specified on DEAL_SKU_TEMP. Costcalc simply takes these item/supplier/origin country/location/active date records, calculates the cost for this combination and inserts the result into FUTURE_COST, along with a reset date for the item/supplier/country/location record if it has one. A reset date would be a deal closing that caused the cost change in the first place. This reset record on FUTURE_COST is simply the item/supplier/origin country/location/close date and the cost. This design has a few inherent problems. Some deals have no close dates and item reclassifications also have no close dates but potentially may change the cost of the item since different deals may apply due to the new merchandise hierarchy classification. Therefore a record on DEAL_SKU_TEMP with no close date may affect records on FUTURE_COST that have an active date later than the record on DEAL_SKU_TEMP. The solution is to move records from FUTURE_COST that are potentially affected by DEAL_SKU_TEMP records back to DEAL_SKU_TEMP, thus guaranteeing that they get recalculated and the correct price will be set as they are re-inserted into FUTURE_COST by costcalc. This process of checking for affected records and moving them into DEAL_SKU_TEMP is performed by precostcalc. Also a new table was created called RECLASS_COST_CHG_QUEUE which holds reclassification, cost change, new item-location events for items along with a general record that simply holds an item which needs to be inserted into DEAL_SKU_TEMP (if an item reclassification was cancelled, we still need to send in a record to DEAL_SKU_TEMP to make sure the item’s record on FUTURE_COST is re-calculated).

Case UPCs: Component items of case UPCs are never inserted into DEAL_SKU_TEMP. These items will be processed by costcalc as part of their case UPCs. Therefore if a case UPC component is inserted into RECLASS_COST_CHG_QUEUE, the driving cursor should ignore it and the record should be deleted. Also an item should be approved, transaction level and not a buyer pack to qualify for insertion into DEAL_SKU_TEMP.

Three quick examples of what appears in RECLASS_COST_CHG_QUEUE for a re-class, a cost change, a new item-location record or a general record:

Re-classification: I need an ITEM, -1 for LOCATION, SUPPLIER, ORIGIN_COUNTRY_ID, START_DATE, DIVISION, GROUP_NO, DEPT, CLASS, SUBCLASS, REC_TYPE = ‘R’.

Cost change: I need an ITEM, LOCATION, SUPPLIER, ORIGIN_COUNTRY_ID, START_DATE, REC_TYPE = ‘C’. (For warehouse locations, only insert virtual warehouses!)

New item location: I need an ITEM, LOCATION, SUPPLIER, ORIGIN_COUNTRY_ID, START_DATE, REC_TYPE = ‘N’. (For warehouse locations, only insert virtual warehouses!)

General: Primary key fields only. (These records appear as placeholders for cancelled events. For example, the merchandiser may cancel a future reclassification event in which case the original event’s record on RECLASS_COST_CHG_QUEUE would be updated to have a rec_type of ‘G’ and a process_flag of ‘N’. This results in precostcalc reinserting this event into DEAL_SKU_TEMP, from there costcalc will re-insert the event into FUTURE_COST. Thus external systems that exported the event earlier will see the event again in FUTURE_COST with potentially a new cost and can export again if necessary. Once such a record has been migrated to FUTURE_COST, there is no need to keep it in RECLASS_COST_CHG_QUEUE too. It may be deleted. The same is true for records with a rec_type ‘N’.

Outstanding Design Issues

N/A

Promotion Price Update [prmpcupd]

Design Overview

This new program will update item_loc table with promotion price information. It will update the promotion fields, promo_retail, promo_selling_retail and promo_selling_uom, in the item_loc table with promotion price information when a simple promotion applies the item/location combination. It will also update these promotion fields to null when a promotion ends today. In addition, it will update the item_loc table with any promotion changes, including add promotion items to the extracted promotions, delete promotion items from the extracted promotions and make promotion price changes to the extracted promotions.

This program will run daily in nightly batch cycle and should be run after the prmext.pc.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
promhead	No	Yes	No	No	No
promstore	No	Yes	No	No	No
Promsku	No	Yes	No	No	No
Item_loc	No	No	No	Yes	No
period	No	Yes	No	No	No
dual	No	Yes	No	No	No
Item_master	No	Yes	No	No	No

Stored Procedures / Shared Modules (Maintainability)

PROMOTION_ATTRIB_SQL.GET_PROMO_RETAIL – Returns the promotion retail price for the item/location.

UOM_SQL.CONVERT – converts the values between two UOMs.

Input Specifications

N/A

Output Specifications

N/A

Function Level Description

- Define a structure that will be used to define the driving cursor array.
- Define another structure to be used to define the update array.

Main():

This function should follow standard Retek main function format. It should call init(), process() and final() function.

Init():

This function performs preliminary processing and populates global variables. It will call retek_init function to handle the restart recovery logic and bring back the bookmark string in case of a restart. It will also retrieve the date of today (vdate) and tomorrow (vdate + 1) to be used in retrieving the valid promotions to process.

Process():

This function will select the promotion, promotion store, store promotion start date, store promotion end date, promotion currency code, promotion item, standard unit of measure, promotion item status, promotion item price change type, amount, selling unit of measure, adjust type and the price ends in from the PROMSTORE, PROMHEAD, PROMSKU and ITEM_MASTER table. It will then call process_prom_item_loc to check the item/location relation and retrieve the promotion price. It will also call the update_item_loc function to update the item_loc table with the promotion price. The driving cursor should be similar as follows:

```
SELECT ph.promotion,
                ph.currency_code,
                TO_CHAR(ps.start_date, 'YYYYMMDD'),
                TO_CHAR(ps.end_date, 'YYYYMMDD'),
                ps.store,
                im.item,
                im.standard_uom,
                psku.status,
                psku.change_type,
                NVL(psku.change_amt,0),
                NVL(psku.selling_uom,’’),
                psku.adjust_type,
                NVL(psku.ends_in,0)
   FROM v_restart_store rv,
               promstore ps,
               promhead ph,
               promsku psku,
               Item_master im
  WHERE  ph.promotion = ps.promotion
      AND ph.status in ('E', 'M')            /* all extracted promotions     */
      AND ps.extract_status in ('E', 'M')            /* all extracted promotion stores
*/
      AND psku.promotion = ps.promotion
  AND ps.start_date <= :tomorrow
  AND ps.end_date >= :today
  AND ps.promotion    = psku.promotion
  AND psku.change_type != 'EX'
  AND im.item_level <= im.tran_level
  AND (im.item = psku.item
       OR (im.item_parent = psku.item
           AND (psku.diff_id is null
                OR (psku.diff_id is not null
                    And (psku.diff_id = im.diff_1
                         OR psku.diff_id = im.diff_2
                         OR psku.diff_id = im.diff_3
                         OR psku.diff_id = im.diff_4))))
        OR(im.item_grandparent = psku.item
           AND(psku.diff_id is null
                OR(psku.diff_id is not null
                    AND (psku.diff_id = im.diff_1
                         OR psku.diff_id = im.diff_2
                         OR psku.diff_id = im.diff_3
                         OR psku.diff_id = im.diff_4)))))
 AND rv.driver_value = ps.store
 AND rv.driver_name = :ora_restart_driver_name
 AND rv.num_threads = :ora_restart_num_threads
 AND rv.thread_val  = :ora_restart_thread_val
 AND (ps.promotion > NVL(:ora_restart_promotion, -999) OR
      ((ps.promotion = :ora_restart_promotion) AND
        (ps.store >= :ora_restart_store)))
ORDER BY 1,5;
```

The flow of this function should be as follows:

- Define an array, la_prom_store, to hold the data fetched by the driving cursor.
- Define another array, la_item_loc, to hold the promotion data to be updated to the item_loc table.
- Call function size_prom_array().
- Call function size_item_loc_array().
- Open the driving cursor
- Array fetch the driving cursor to the la_prom_store array for commit_max_ctr records.
- In a for loop, loop through each record in the la_prom_store array
- Call function process_prom_item_loc(). Pass in all the elements in current record to the function, as well as the item_loc array.
- If the current promotion/store combination is different from last one, and the count of the item_loc array is greater than zero, call update_item_loc() to update the item_loc table. Pass in the item_loc array and the count of the records in the array. Note the count needs to be reset after the update.
- Commit and restart/recovery logic.
- Remember to update item_loc table with the last set of records in the item_loc array.

Size_prom_array():

This function will allocate memory for the array la_prom_store to size of commit_max_ctr.

Size_item_loc_array():

This function will allocate memory for the item_loc array to size of commit_max_ctr.

Process_prom_item_loc():

This function should process as follows:

- Define local variables to hold the promo_retail, promo_selling_retail, promo_selling_uom, and rowid Initialize these variables to null.
- Create a cursor c_item_loc to retrieve the promo_retail, promo_selling_retail, promo_selling_uom and rowid from item_loc where status is 'A'ctive and the item, location match the passed in item and location.
- If no record found, return true. If error found, return fatal.
- If the end of store promotion date is today, call populate_item_loc_array function. Pass in the item_loc array and the local variables item,location promo_retail, promo_selling_retail, promo_selling_uom and rowid. Return whatever is returned from the populate_item_loc function.
- Check the store Promotion start date:

If the start date is less than or equal to tomorrow, and the promsku.status is ‘DI’ (Deleted Item) or ‘DP’ (Delete Processed), check

- if the promo_selling_unit_retail is null. If it is null, stop further processing and the function should return true.
- if the promo_selling_unit_retail is not null, call populate_item_loc_array function. Pass in the item_loc array and the local variables item, location, promo_retail, promo_selling_retail, promo_selling_uom and rowid. Return whatever is returned from the populate_item_loc function.

Otherwise,

- Call get_prom_retail() to retrieve the promotion price. Pass in the current records in the promo_store array and the local variable promo_selling_retail. Return false if the function call failed.
- Compare the promo_selling_retail and the promo_selling_uom obtained in the get_prom_retail with the promo_selling_retail and promo_uom in the item_loc table (retrieved from cursor c_item_loc). If they are same, stop further processing and return true.
- Call calc_std_retail() to convert the promotion retail to standard retail. Pass in the selling_uom, standard_uom, promo_selling_retail, promo_retail, item and location. Return false if the function call failed.
- Call populate_item_loc_array(). Pass in the item_loc array and the local variables item, location, promo_retail, promo_selling_retail, promo_selling_uom and rowid. Return whatever is returned from the populate_item_loc function.

Populate_item_loc_array():

This function will process as follows:

- Check if the count of the total records in the item_loc array plus 1 is greater than the size of the item_loc array. If it is exceed the array size, call resize_item_loc array().
- Copy the promo_retail, promo_selling_retail, promo_selling_uom and rowid to the item_loc array. Populate the item_loc array’s last_update_datetime and last_update_id with the SYSDATE and UESER, respectively.
- Return ture.

Resize_item_loc_array():

This function will allocate additional max_commit_ctr memory for the item_loc array.

Get_prom_retail():

This function will call stored procedure PROMOTION_ATTRIB_SQL.GET_PROMO_RETAIL to retrieve the promo_selling_retail.

Calc_std_retail():

This function will first retrieve the convert factor from a selling_uom to standard_uom. Then calculate the promotion retail per standard unit of measure. It should process as the steps descript below:

- Call stored procedure UOM_SQL.CONVERT to find the convert factor. Pass 1 to the from_value, pass selling_uom to the from_uom and pass standard_uom to the to_uom.
- Set promo_retail = promo_selling_retail / convert factor.

Update_item_loc():

This function will do an array update against the item_loc table using the item_loc array passed where the records rowid in the item_loc table equal the rowids in the item_loc array.

Final():

This function will call retek_close to perform the restart/recovery closing logic, as well as the last commit of the database changes.

Scheduling Considerations

Processing Cycle: Phase 3

Pro-Processing: Prmext.pc

Post-Processing:

Threading Scheme: store

Locking Strategy

N/A

Restart/Recovery

The logic unit of work is promotion/location.

Performance Considerations

N/A.

Security Considerations

N/A.

Design Assumptions

This program will update the item_loc table for simple promotions, no matter if the system_options.multi_promo_ind flag is on or off.

When a promotion is changed, for example an item is added to the promotion, the promotion item status will be updated or removed after the execution of prmext.pc. In order to catch the promotion changes, this program will do a full scan for all valid promotions, and update the item_loc table with new promotion prices and the changed promotion/item/location prices.

Outstanding Design Issues

N/A

Stockout Download [soutdnld]

Design Overview

Retek Demand Forecasting (RDF) requires notification when an item/store's stock on hand is at zero or below. This program will loop through the item/store tables and output any item/store combination that has a stock out condition to an output file. This output file will then be sent to RDF.

The logical unit of work (LUW) for this program is item/store.

Stored Procedures / Shared Modules (Maintainability)

N/A

Input Specifications

This program outputs three fields: date, item, and store to RDF. The fields should be sent for each store/item combination that has a stock-on-hand less than or equal to zero. The date sent will always be the vdate from the PERIOD table. The item and store come from the ITEM_MASTER table. This program will not look at packs. All items must be forecastable to be considered by this program. The forecastable indicator is held on the ITEM_MASTER table.

RDF requires that the output files generated by this program be grouped by domain number. To accommodate this requirement, soutdnld.pc should be threaded by domain. Since threads are determined by the value of the domain ID, the restart_program_status table should contain a row for each domain ID. The thread value of the domain ID should be used as the thread value on this table. The total number of domains/number of threads should be equal to the number of rows on the restart_program_status table. This value must be entered into the restart_control table num_threads field. Note that anytime a new domain is created that an additional row should be added to the restart_program_status table with the thread value equal to the domain ID and the restart_control table num_threads field must be incremented to equal the total number of domains.

Domains can be held in RMS at the dept, class, or the subclass level. The SYSTEM_OPTIONS.DOMAIN_LEVEL holds the domain level that is being used. This dictates the program will need three different driving cursors. Which one will be used depends on the SYSTEM_OPTIONS.DOMAIN_LEVEL. The cursors will all be identical except one will join to DOMAIN_DEPT, one will join to DOMAIN_CLASS, and one will join to DOMAIN_SUBCLASS.

When the SYSTEM_OPTIONS.DOMAIN_LEVEL is 'D', the following should be used as the driving cursor:

```
SELECT item_loc_soh.item,
    item_loc_soh.loc
 FROM item_loc_soh,
    item_master,
    domain_dept dd
WHERE item_loc_soh.stock_on_hand <= 0
```

```
     AND item_loc_soh.loc_type = 'S'
     AND item_master.forecast_ind = 'Y'
     AND item_loc_soh.item = item_master.item
     AND dd.dept = item_master.dept
     AND dd.domain_id = :ps_thread_val
     AND (item_loc_soh.item > NVL(:ps_restart_item, -999) OR
          (item_loc_soh.item = :ps_restart_item AND
           (item_loc_soh.loc > :ps_restart_loc)))
  ORDER BY 1,2;
```

When the SYSTEM_OPTIONS.DOMAIN_LEVEL is ‘C’, the following should be used as the driving cursor:

```
   SELECT item_loc_soh.item,
          item_loc_soh.loc
     FROM item_loc_soh,
          item_master,
          domain_class dc
    WHERE item_loc_soh.stock_on_hand <= 0
     AND item_loc_soh.loc_type = 'S'
     AND item_master.forecast_ind = 'Y'
     AND item_loc_soh.item = item_master.item
     AND dc.dept = item_master.dept
     AND dc.class = item_master.class
     AND dc.domain_id = :ps_thread_val
     AND (item_loc_soh.item > NVL(:ps_restart_item, -999) OR
          (item_loc_soh.item = :ps_restart_item AND
           (item_loc_soh.loc > :ps_restart_loc)))
  ORDER BY 1,2;
```

When the SYSTEM_OPTIONS.DOMAIN_LEVEL is ‘S’, the following should be used as the driving cursor:

```
   SELECT item_loc_soh.item,
          item_loc_soh.loc
     FROM item_loc_soh,
          item_master,
          domain_subclass ds
    WHERE item_loc_soh.stock_on_hand <= 0
    AND item_loc_soh.loc_type = 'S'
    AND item_master.forecast_ind = 'Y'
    AND item_loc_soh.item = item_master.item
    AND ds.dept = item_master.dept
    AND ds.class = item_master.class
    AND ds.subclass = item_master.subclass
    AND ds.domain_id = :ps_thread_val
    AND (item_loc_soh.item > NVL(:ps_restart_item, -999) OR
        (item_loc_soh.item = :ps_restart_item AND
         (item_loc_soh.loc > :ps_restart_loc)))
ORDER BY 1,2;
```

Output Specifications

soutnn.dat – This is the file that will be created where *nn* is the thread (domain) number.

Example record for a date of 20010309, a store of 1000000000, and a item of 101742484.

```
12345678123456789012345678901234567890123456789012345
2001030910000000000           101742484
```

Record Name	Field Name	Field Type	Default Value	Description
	Date	Varchar2(8)	Period.vdate	The date of the stockout in YYYYMMDD format.
	Store	Varchar2(20)		The store at which the sku encountered the stockout – left justified with trailing blanks.
	Item	Varchar2(25)		The item that encountered the stockout – left justified with trailing blanks.

Function Level Description

Main()
The standard RMS main function. Calls init(), process(), and final().

Init()
The restart recovery is initialized by calling retek_init(), and the output file is set up. The file should be named soutnn.dat, where nn is the thread number. The C_get_vdate_and_domain_level cursor is called to get the vdate from the period table and the domain_level from system_options.

Create_foremat_strings()
This function populates a format string. The format string is used when writing lost sales to the output file.

Process()
Initializes the fetch struct, then calls the create_format_strings() funciton. Uses the domain_level fetched in Init() to call either Process_dept(), Process_class(), or Process_subclass(). Finally calls Free_array().

Write_lost_sales()
A pointer of the fetch struct and the current number of records to print are input. The date, item, and store are then written to the output file. The format string defined by create_format_string() is used when doing the file write.

Process_dept()
A pointer to the fetch struct is passed in. Contains the C_dept_domain_level cursor for processing when SYSTEM_OPTIONS.DOMAIN_LEVEL = 'D'. Once the C_dept_domain_level cursor is opened, a While(1) loop is entered. Inside the While(1) loop, an array fetch is performed, and a break_flag variable is set to one if a fetch returns NO_DATA_FOUND. The number of records fetched is then passed, along with the struct containing the fetched data, to the Write_lost_sales() function. After the Write_lost_sales() function is called, the break_flag is checked, and the While(1) loop is broken if the break_flag != 0.

Process_class()
A pointer to the fetch struct is passed in. Contains the C_class_domain_level cursor for processing when SYSTEM_OPTIONS.DOMAIN_LEVEL = 'C'. Once the C_class_domain_level cursor is opened, a While(1) loop is entered. Inside the While(1) loop, an array fetch is performed, and a break_flag variable is set to one if a fetch returns NO_DATA_FOUND. The number of records fetched is then passed, along with the struct containing the fetched data, to the Write_lost_sales() function. After the Write_lost_sales() function is called, the break_flag is checked, and the While(1) loop is broken if the break_flag != 0.

Process_subclass()
A pointer to the fetch struct is passed in. Contains the C_subclass_domain_level cursor for processing when SYSTEM_OPTIONS.DOMAIN_LEVEL = 'S'. Once the C_subclass_domain_level cursor is opened, a While(1) loop is entered. Inside the While(1) loop, an array fetch is performed, and a break_flag variable is set to one if a fetch returns NO_DATA_FOUND. The number of records fetched is then passed, along with the struct containing the fetched data, to the Write_lost_sales() function. After the Write_lost_sales() function is called, the break_flag is checked, and the While(1) loop is broken if the break_flag != 0.

Init_array()
The initialized fetch struct is passed in. Allocates array space for the fetch struct arrays, using calloc. The array sizes are set to the value of the COMMIT_MAX_COUNTER.

Free_array()
Frees the array space that was allocated in Init_array().

Final()
Calls Retek_close().

Scheduling Considerations

Phase 4 daily. Any processing that updates the stock levels should be completed before this program runs.

Locking Strategy

N/A

Restart/Recovery

This program should use restart recovery. The LUW for this program is each unique sku store combination.

Performance Considerations

N/A.

Security Considerations

N/A.

Design Assumptions

N/A

Outstanding Design Issues

N/A

References

N/A

Appendix

N/A

Item–Location Ticket Output File [tcktdnld]

Design Overview

This program will create an output file containing all of the information to be printed on a ticket or label for a particular ITEM/location. This program is driven by the “requests” for tickets that exist on the TICKET_REQUEST table. Information to be printed on the ticket is then retrieved based on the ITEM, location and the ticket type requested. The details, which should be printed on each type of ticket, are kept on the TICKET_TYPE_DETAIL table. Specific details, which will be written to the output file, are taken from the various item tables (i.e. ITEM short description from ITEM_MASTER, retail price from ITEM_ZONE_PRICE).

Scheduling Contraints

Processing Cycle:	Ad Hoc (Daily)
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

Restartability will exist implicitly within this program. Because records will be deleted after they are selected, no explicit code is needed to restart in the event of a failure.

The lack volume of data processed by this program, in addition to the lack of an appropriate threading mechanism, negates the need for Retek multi-threading capabilities.

Driving Cursor:

```
SELECT tr.ticket_type_id,
    tr.item,
    tr.quantity,
    tr.loc_type,
    tr.location,
    tr.country_of_origin,
    tr.unit_retail,
    tr.multi_units,
    tr.multi_unit_retail,
    ROWIDTOCHAR(tr.rowid)
 FROM ticket_request tr
ORDER BY 1, 2, 5;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

init() –

Functional details:

This function should initialize the restart/recovery process. The output file should be opened, and if it is not a "restart", then file header information should be written. The system vdate is selected for the file create date used in the output file header. The format_buffer function should be called to format output strings. The size_arrays function should be called to size the fetch and delete arrays.

Technical details:

The output file should be written in the style necessary for Retek restart/recovery. That is, a temp file should be opened and initialized, and the final file should only be written to when the restart/recovery commit logic is called (using restart_file_write).

process() –

Functional Details:

This function should write transaction records to the output file for each item/ticket type/location combination on the ticket_request table. For each record a transaction header should be written to the output file and the ticket_item function should be called to write the detail items for the details associated with the ticket type. If the item is a pack item, however, the ticket_pack function should be called to first write component item records (the ticket_item function will be called within the ticket_pack function for each component item.). After each record from the driving cursor is processed, it will be deleted from the ticket_request table. Finally, when all of the records have been processed from the table, a file trailer should be written to the output file.

Technical Details:

The function should fetch records from the driving cursor into arrays. The arrays should be sized to match the value of the maximum commit counter on the restart_control table. Once the records are fetched, each record should be processed in a for-loop. After all of the records have been processed in the for loop, the records should be array deleted from the ticket_request table by rowid, and the restart_commit logic should be called. Output file line counters, transaction counters, etc. should be saved into the application image array string that is passed to the restart_control function.

ticket_pack()

This function will be called from process if the item on the ticket_request table is a pack item. This function should fetch all of the component items in the pack, along with pack quantity information, and write a pack record for each component. Further the ticket_item function should be called for each component.

ticket_item()

This function should select all of the records from the ticket_type_detail table with the ticket_type from the ticket_request record. Detail records should be written out to the output file for each detail record retrieved. Either item information or attribute information should be written to the output file. If the ticket item is to be written (fetched ticket item is not null) get_ticket_item is called to retrieve this appropriate intormation. If the attribute information is to be written (fetched attribute column is not null) then a function should be called to get the appropriate attribute information (get_UDA).

get_ticket_item()–

This function retrieves the database information which corresponds to the requested ticket item, according to the table below.

TICKET ITEM	OUTPUT FILE VALUE
UOM	Price per unit of measure from item_master.
ITEM	Retek ITEM value
ITDS	ITEM description (from item_master)
ITSD	ITEM short description from the item_master table
VAR	The primary variant (ref_item) from the item_master table
DIF1	Diff_1 value from item_master
DIF2	Diff_2 value from item_master
DIF3	Diff_3 value from item_master
DIF4	Diff_4 value from item_master
WGHT	Case weight from item_supp_country_dim table
DEPT	Department from item_master & department name from deps table
CLAS	Class value from item_master table & class name from class table
SBCL	Subclass from item_master table & subclass name from subclass table
RTPC	Selling retail price from driving cursor (if available), otherwise from item_zone_price for item/store (use base zone value for warehouses).
SRTP	Suggested retail price (mfg_rec_retail) from item_master
MUPC	Multi-units and multi-unit retail from driving cursor (if available), otherwise from item_zone_price for item/store (use base zone value for warehouses)
SUPR	Supplier from ordhead for most recent PO for the SKU.

TICKET ITEM	OUTPUT FILE VALUE
SUP1	Supplier diff_1, from item_supplier
SUP2	Supplier diff_2, from item_supplier.
SUP3	Supplier diff_3, from item_supplier
SUP4	Supplier diff_4, from item_supplier
STRE	Store from driving cursor
WHSE	Warehouse from driving cursor
COOG	Country of origin from driving cursor if available, else from the last PO (see supplier).

get_UDA()

This function should fetch the user defined attribute (UDA) value and description assocated with the attribute value selected from the ticket detail table. The UDA description will be selected for the UDA and the ITEM from either the UDA_item_lov and the UDA_value tables, the UDA_item_ff table or the UDA_item_date table. The UDA value will be written to the output file in the "value" location of the detail line.

final()

Retek restart/recovery process will be closed by calling the internal API function, and all appropriate output files will be close and temp files will be removed.

I/O Specification

Output File:

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence	Number(10)		Line number of the current file
	File Type Definition	Char(4)	TCKT	Identifies file as 'Print Ticket Requests'
	File Create Date	Date	create date	date file was written by external system
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type
	File Line Sequence	Number(10)		Line number of the current file

Record Name	Field Name	Field Type	Default Value	Description
	ITEM	Char(25)		ID number of the transaction level, non-pack item or the pack item
	Ticket Type	Char(4)		ID which indicates the ticket type to be printed
	Location Type	Char(1)	S - Store W – Warehouse	Identifies the type of location for which tickets will be printed
	Location	Char (10)		number of the store or warehouse for which tickets will be printed
	Quantity	Number(12,4)		the quantity of tickets to be printed
Transaction Component	File Type Record Descriptor	Char(5)	TCOMP	Identifies file record type
	File Line Sequence	Number(10)		Line number of the current file
	ITEM	Char(25)		ID number of the ITEM
	Quantity	Number(12,4)		Quantity of the component ITEM as part of the whole; if ITEM on the header record is a transaction level ITEM, the value in this field will be 1.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence	Number(10)		Line number of the current file
	Detail Sequence Number	Number(10)		Sequential number assigned to the detail records
	Ticket Item	Char(4)		ID indicating the detail to be printed on the ticket
	Attribute Description	Char(40)		Description of the attribute (from the UDA Table)

Record Name	Field Name	Field Type	Default Value	Description
	Value	Char(100)		Detail to be printed on the ticket (i.e. REF_ITEM, Department Number, ITEM description)
	Supplement	Char(300)		Supplemental description to the Value (i.e. Department Name)
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Sequence	Number(10)		Line number of the current file
	Transaction Detail Line Count	Number(6)	sum of detail lines	sum of the detail lines within a transaction
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence	Number(10)		Line number of the current file

Technical Issues

The program could be sped up by outer joining ticket_type_detail into the driving cursor, and avoiding the c_ttd cursor which must be opened for each record in our fetch array.

VAT–Rate Maintenance [vatdlxpl]

Design Overview

Value Added Tax (VAT) is a tax imposed by some governments on goods that have realized an increase in value.

As with price zones and cost zones, individual stores belong to a particular VAT region. VAT regions are areas that contain VAT code and VAT rates information that applies to stores in that region and are based on VAT rates defined by the government. VAT codes can be set up by Retek users to identify a particular VAT rate percentage. Only one VAT rate can be active for any one VAT code.

VAT rates are stored at the ITEM level in Retek on the VAT_ITEM table. On this table, records with a past active date serve as an audit trail to track what VAT code and VAT rate a particular ITEM has had or currently has. The VAT_ITEM record with the most recent active date holds the ITEM's current VAT code. The VAT rate related to a particular VAT code is found on the VAT_CODE_RATES table. Records on VAT_ITEM with a future active date functionally represent a pending change in the VAT code for that particular ITEM, effective on that date and reflecting the corresponding VAT rate associated with that VAT code on the VAT_CODE_RATES table.

Records on the VAT_CODE_RATES table functionally represent a change in the VAT rate for a particular VAT code. When a record exists on this table with an active date of tomorrow, this program updates the VAT rate on any future VAT_ITEM records associated with that VAT code to the new VAT rate defined in the VAT_CODE_RATES table. Also, if the latest VAT_ITEM record for any ITEM contains the VAT code that is changing, a new VAT_ITEM record is inserted for that ITEM with an active date of tomorrow and the new VAT rate associated with the VAT code.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
VAT_CODE_RATES	Yes	Yes	No	No	No
VAT_ITEM	Yes	Yes	Yes	Yes	No

Scheduling Contraints

Processing Cycle:	Phase I
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

The logical unit of work will be based on VAT code. Restart/recovery will be based on VAT code. It is recommended that records be committed after a set of 10,000 rows has been processed.

Program Flow

main()

- **init()** — Retrieve today's date (VDATE) off database table.
- **process()** — Retrieve the vat_rate records from the vat_code_rates table where the active date is equal to today.

  For all SKUs on the vat_ sku table that are associated with the vat_code whose vat_rate changed, insert/update the record in the vat_sku table with new vat_rate.
- **final()** — Commit changes to the database.

Shared Modules

N/A

Function Level Description

init():

Retrieve tomorrow's date off the period table.

process():

This function will retrieve all records off of the VAT_CODE_RATES table with an active date of tomorrow. If any records exist on the VAT_ITEM table with an active date on or after tomorrow for the VAT code fetched off of the VAT_CODE_RATES table, their VAT rate is updated with the VAT rate fetched off of the VAT_CODE_RATES table. Also, if an ITEM is currently assigned the VAT code with the changing VAT rate, a new record is inserted into VAT_ITEM for the new VAT code/VAT rate combination with an active date of tomorrow.

update_vat_item():

This function updates any future VAT_ITEM records with the new VAT rate fetched off the VAT_CODE_RATES table. It utilizes array processing to handle the update.

insert_vat_item():

Inserts a new record into the VAT_ITEM table for any ITEMs that are currently associated with the VAT code fetched from the VAT_CODE_RATES table. A new record is inserted into VAT_ITEM for the new VAT code/VAT rate combination with an active date of tomorrow.

Insert_pos_mods()

This function is called after the update_vat_item function. It inserts a new record with a tran type of 20 into the POS_MODS table. The item, store, dept, class, subclass, vat code, vat rate, and class_vat_ind fields are populated for all approved, transaction-level items found on the VAT_ITEM table and their children with tomorrows date and vat code, at active stores.

Technical Issues

This program implements the use of array processing to handle the inserts/updates of the VAT_ITEM table.

Wastage Adjustment [wasteadj]

Design Overview

This program will reduce inventory of spoilage type wastage items to account for natural wastage that occurs over the shelf life of the product. Only items with spoilage type wastage will be affected by this program. Sales type wastage will be accounted for at the time of sale.

Spoilage type wastage is due to the natural loss of a product over its shelf life. For example, as sausage sits on the shelf it loses a certain percentage of its weight each day due to evaporation of the water that is in the sausage. Therefore, a sausage that weighs one pound today may only weight .9 pounds tomorrow due to the loss of water from the sausage. This new batch program will reduce the physical on hand inventory for a store based on the daily wastage percentage which will be entered by the retailer at the item location level:

New Stock on Hand = Actual Stock on Hand * (1 – Daily Wastage %)

When stock is reduced, Merchandising will process these records just as any other stock adjustment.

By automatically reducing stock on hand based on the wastage percentage and the shelf life of an item, the on hand inventory in Merchandising will be more accurate than if these adjustments were not made.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ITEM_MASTER	No	Yes	No	No	No
ITEM_LOC_SOH	No	Yes	No	Yes	No
ITEM_LOC	No	Yes	No	No	No
INV_ADJ	No	No	Yes	No	No
TRAN_DATA	No	No	Yes	No	No
PERIOD	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	Yes	No
CLASS	No	Yes	No	No	No

Scheduling Contraints

Processing Cycle: Phase III (Before stock ledger processing)

Scheduling Diagram:

Pre-Processing: Run this program before the stock ledger roll up programs to make sure that the stock adjustments taken during the current day are credited to the appropriate day.

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

Restart recovery is based on last item processed. The program will commit only when the commit max counter is reached.

Program Flow

N/A

Shared Modules

N/A

Function Level Description

init

- Initialize restart variables.
- Allocate memory to arrays for array processing
- Collect system information

process

- Fetch ITEMs with wastage type spoilage ('SP')
- Retrieve single ITEM from fetch array for processing
- If a the ITEM is a transaction level item:
- retrieve appropriate information from ITEM_LOC and ITEM_LOC_SOH
- add information to insert/update arrays

create_inv_adj

- Adds a record to the inv_adj insert array

create_tran_data

- Adds a record to the tran_data insert array

get_vat_rate

- gets vat_rate based on item, dept and location

set_tran_data_retail

- determines whether vat needs to be added, stripped or neither to total_retail (Calls library function ADD_VAT or REMOVE_VAT in common.h)

update_stock

- Performs array updates of ITEM_LOC_SOH records

write_arrays

- Performs array inserts to inv_adj and tran_data

final

- Closes restart recovery

I/O Specification

N/A

Technical Issues

N/A

Customization Issues

N/A

ReSA RTLOG interface file layout

The following table shows the layout of the Retek Sales Audit RTLOG interface file for RMS.

Record Name	Field Name	Field Type	Default Value	Description	Required
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.	Yes
	File Type Definition	Char(4)	POSU	Identifies file as 'POS Upload'	Yes
	File Create Date	Char(14)	create date	date file was written by external system	Yes
	Location Number	Number(10)	specified by external system	Store or warehouse identifier	Yes
	Vat include indicator	Char(1)		Determines whether or not the store stores values including vat. Not required but populated by Retek sales audit	Yes
	Vat region	Number(4)		Vat region the given location is in. Not required but populated by Retek sales audit	Yes
	Currency code	Char(3)		Currency of the given location. Not required but populated by Retek sales audit	Yes
	Currency retail decimals	Number(1)		Number of decimals supported by given currency for retails. Not required but populated by Retek sales audit	Yes

Record Name	Field Name	Field Type	Default Value	Description	Required
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.	Yes
	Business Date of Transaction	Char(14)	business date	Business date that the sale/return transaction was processed at the POS	Yes
	Item Type	Char(3)	'ITM' or 'REF'	item type will be represented as an ITEM or REF_ITEM	Yes
	Item	Char(25)	item identifier		Yes
	Dept	Number(4)	Item's dept	Dept of item sold or returned. Not required but populated by Retek sales audit	Yes
	Class	Number(4)	Item's class	Class of item sold or returned. Not required but populated by Retek sales audit	Yes
	Subclass	Number(4)	Item's subclass	Subclass of item sold or returned. Not required but populated by Retek sales audit	Yes
	Pack Indicator	Char(1)	'Y' – pack_item 'N' – non_pack	Indicates if the item is a pack.	Yes
	Item level	Char(1)	Level 3- item grandparent Level 2 – item parent Level 1 - item	Indicates which of the three levels the item resides.	Yes

Record Name	Field Name	Field Type	Default Value	Description	Required
	Transaction level	Char(1)	Level 1,2, or 3	Indicates which level that transactions occur for the item's group.	Yes
	Wastage Type	Char(6)	Item's wastage type	Wastage type of item sold or returned. Not required but populated by Retek sales audit	Yes
	Wastage Percent	Number(12)	Item's wastage percent	Wastage percent of item sold or returned with 4 implied decimal places. Not required but populated by Retek sales audit	Yes
	Transaction Type	Char(1)	'S' – sales 'R' - return	Transaction type code to specify whether transaction is a sale or a return	Yes
	Drop Ship Ind	Char(1)	'Y' or 'N'	Indicates whether item is part of a drop shipment	No
	Total Sales Quantity	Number(12)		Number of units sold at a particular location with 4 implied decimal places.	Yes
	Selling UOM	Char(4)		UOM at which this item was sold	No
	Sales Sign	Char(1)	'P' - positive 'N' - negative	Determines if the Total Sales Quantity and Total Sales Value are positive or negative.	Yes

Record Name	Field Name	Field Type	Default Value	Description	Required
	Total Sales Value	Number(20)		Sales value, net sales value of goods sold/returned with 4 implied decimal places.	Yes
	Last Modified Date	Char(14)		For VBO future use	
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies transaction record type	
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.	Yes
	Promotional Tran Type	Char(6)	promotion type – valid values see code_detail table.	code for promotional type from code_detail, code_type = 'PRMT'	Yes
	Promotion Number	Number(10)	promotion number	promotion number from the RMS	No
	Sales Quantity	Number(12)		number of units sold in this prom type with 4 implied decimal places.	Yes
	Sales Value	Number(20)		value of units sold in this prom type with 4 implied decimal places.	Yes
	Discount Value	Number(20)		Value of discount given in this prom type with 4 implied decimal places.	Yes
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.	Yes
	Transaction Count	Number(6)	specified by external system	Number of TDETL records in this transaction set	Yes

Record Name	Field Name	Field Type	Default Value	Description	Required
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.	Yes
	File Record Counter	Number(10)		Number of records/transactions processed in current file (only records between head & tail)	Yes