Retek Retail passion. Innovative solutions.

Retek® Merchandising System™ 11.0.1

Operations Guide Addendum

Corporate Headquarters:

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

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

Fax:
+1 612 587 5100

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

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

Fax:
+44 (0)20 7563 46 10

Customer Support

Customer Support hours

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

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

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

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

Australia	+1 800 555 923 (AU-Telstra) or +1 800 000 562 (AU-Optus)
France	0800 90 91 66
Hong Kong	800 96 4262
Korea	00 308 13 1342
United Kingdom	0800 917 2863
United States	+1 800 61 RETEK or 800 617 3835

Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

When contacting Customer Support, please provide:

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

Contents

Chapter 1 – Publication And Subscription Designs

Item Loc Publication API.doc

Business Overview:

RMS defines Item-Location relationships. In Retek XI, these will be published from RMS to ISO and RPM via the RIB. The existing RMS Item Location publisher will be adapted to achieve this publication.

To support ISO, RMS is adding a Store Price indicator to the Item-Location relationship. This indicator specifies whether or not that Store can mark the price of the Item down. Retek Integration will add this attribute to the Item-Location message.

RMS will also support the ISO Initial Retail requirement. While all subsequent price changes will be taken from RPM, ISO needs RMS to send a starting set of Retails for each Item-Location. To meet this requirement, Retek Integration will add Retail fields to the Item-Location message. These fields will be published upon creation; subsequent updates to these Retail fields, however, will not trigger an update message.

In addition to modifications for ISO requirements, the Item-Location publisher will be modified to publish warehouses as well as stores. This is need for RPM purposes.

Functionality Checklist:

Description	RMS	RIB
RMS must publish item loc information		
Create new Publisher	X	X

Form Impact

None.

Business Object Records

None.

Package Impact

- Item Loc publishing – store price initialized the publishing of store_price on Item_LOC
- Item Loc publishing – store price ind to verify the initial publishing of pricing info (unit retail, selling unit retail and uom) on ITEM_LOC.
- Item Loc publishing – pricing, make an update of the pricing info on ITEM_LOC – should NOT be published.

No Change.

Package name: RMSMFM_ITEMLOC

Spec file name: rmsmfm_itemlocs.pls

Body file name: rmsmfm_itemlocb.pls

Package Specification – Global Variables

FAMILY	CONSTANT	VARCHAR2(64)	‘ItemLoc’;
ITEMLOC_ADD	CONSTANT	VARCHAR2(20)	'ItemLocCre';
ITEMLOC_UPD	CONSTANT	VARCHAR2(20)	'ItemLocMod';
ITEMLOC_DEL	CONSTANT	VARCHAR2(20)	'ItemLocDel';
REPL_UPD	CONSTANT	VARCHAR2(20)	'ItemLocReplMod';

Function Level Description – ADDTOQ

```
Function:   ADDTOQ
(O_error_message        OUT         VARCHAR2,
I_message_type          IN          ITEMLOC_MFQUEUE.MESSAGE_TYPE%TYPE,
I_itemloc_record        IN          ITEM_LOC%ROWTYPE,
I_prim_repl_supplier    IN          REPL_ITEM_LOC.PRIMARY_REPL_SUPPLIER%TYPE,
I_repl_method           IN          REPL_ITEM_LOC.REPL_METHOD%TYPE,
I_reject_store_ord_ind  IN          REPL_ITEM_LOC.REJECT_STORE_ORD_IND%TYPE
I_next_delivery_date    IN          REPL_ITEM_LOC.NEXT_DELIVERY_DATE%TYPE);
```

This will call the API_LIBRARY.GET_RIB_SETTINGS if the LP_num_threads is NULL and insert the family record into ITEMLOC_MFQUEUE table. The call for HASH_ITEM will insert the I_itemloc_record.item information into ITEMLOC_MFQUEUE table.

Function Level Description – GETNXT

```
Procedure: GETNXT
(O_status_code          OUT             VARCHAR2,
O_error_msg             OUT             VARCHAR2,
O_message_type          OUT             VARCHAR2,
O_message               OUT             RIB_OBJECT,
O_bus_obj_id            OUT             RIB_BUSOBJID_TBL,
O_routing_info          OUT             RIB_ROUTINGINFO_TBL,
I_num_threads           IN              NUMBER DEFAULT 1,
I_thread_val            IN              NUMBER DEFAULT 1);
```

Make sure to initialize LP_error_status to API_CODES.HOSPITAL at the beginning of GETNXT.

The RIB calls GETNXT to get messages. The driving cursor will query for unpublished records on the ITEMLOC_MFQUEUE table (PUB_STATUS = ‘U’).

Since ITEMLOC records should not be published before ITEM records, include a clause in the driving cursor that checks for ITEM CREATE messages on the ITEM_MFQUEUE table. The ITEMLOC_MFQUEUE record should not be selected from the driving cursor if the ITEM CREATE message still exists on ITEM_MFQUEUE. Also, ITEMLOC_MFQUEUE cleanup should be included in ITEM_MFQUEUE cleanup. When the item publisher RMSMFM_ITEMS encounters a DELETE message for an item that has never been published, it deletes all records for the item from the ITEM_MFQUEUE table. This is done in the program unit CLEAN_QUEUE. CLEAN_QUEUE should now also delete from ITEMLOC_MFQUEUE when a DELETE message for a non-published item is encountered.

After retrieving a record from the queue table, GETNXT should check for records on the queue with a status of ‘H’ospital. If there are any such records for the current business object, GETNXT should raise an exception to send the current message to the Hospital.

The information from the ITEMLOC_MFQUEUE table is passed to PROCESS_QUEUE_RECORD. PROCESS_QUEUE_RECORD will build the Oracle Object message to pass back to the RIB. If PROCESS_QUEUE_RECORD does not run successfully, GETNXT should raise an exception.

If any exception is raised in GETNXT, including the exception raised by an unsuccessful call to PROCESS_QUEUE_RECORD, HANDLE_ERRORS should be called.

Function Level Description – PUB_RETRY

```
Procedure: PUB_RETRY
(O_status_code      OUT         VARCHAR2,
O_error_msg         OUT         VARCHAR2,
O_message           OUT         RIB_OBJECT,
O_message_type      IN  OUT     VARCHAR2,
O_bus_obj_id        IN  OUT     RIB_BUSOBJID_TBL,
O_routing_info      IN  OUT     RIB_ROUTINGINFO_TBL,
I_REF_OBJECT        IN          RIB_OBJECT);
```

Same as GETNXT except:

The record on ITEMLOC_MFQUEUE must match the passed in sequence number (contained in the ROUTING_INFO).

Function Level Description – PROCESS_QUEUE_RECORD (local)

```
Procedure: PROCESS_QUEUE_RECORD
(O_error_message    OUT         VARCHAR2,
O_message           IN  OUT     nocopy RIB_OBJECT,
O_routing_info      IN  OUT     nocopy RIB_ROUTINGINFO_TBL,
O_bus_obj_id        IN  OUT     nocopy RIB_BUSOBJID_TBL,
O_message_type      IN  OUT     VARCHAR2,
I_item              IN          ITEMLOC_MFQUEUE.ITEM%TYPE);
```

This function controls the building of Oracle Objects given the business transaction’s key values and a message type. It contains all of the shared processing between GETNXT and PUB_RETRY.

If the record from ITEMLOC_MFQUEUE table is an add or update (ITEMLOC_ADD, ITEMLOC_UPD)

- Call BUILD_DETAIL_OBJECTS to build the Oracle Object to publish to the RIB. This will also take care of any ITEMLOC_MFQUEUE deletes and ROUTING_INFO logic.

If the record from ITEMLOC_MFQUEUE table is a delete (ITEMLOC_DEL)

- Call BUILD_DETAIL_DELETE_OBJECTS to build the Oracle Object to publish to the RIB. This will also take care of any ITEMLOC_MFQUEUE deletes and the ROUTING_INFO logic.

Function Level Description – BUILD_DETAIL_OBJECTS (local)

```
Procedure: BUILD_DETAIL_OBJECTS
( O_error_msg       IN OUT   VARCHAR2,
O_ilphys_tbl        IN OUT   nocopy RIB_ITEMLOCPHYS_TBL,
O_routing_info      IN OUT   nocopy RIB_ROUTINGINFO_TBL,
I_message_type      IN       ITEMLOC_MFQUEUE.MESSAGE_TYPE%TYPE,
I_item              IN       ITEMLOC_MFQUEUE.ITEM%TYPE,
I_max_details       IN       rib_settings.max_details_to_publish%TYPE);
```

The function is responsible for the Oracle Object used for a DESC message (inserts and updates.) It adds as many mfqueue records to the message as it can given the passed in message type and business object keys.

- Select all records on the ITEMLOC_MFQUEUE that are for the same item. Fetch the records in order of seq_no on the MFQUEUE table. Fetch the records into a table using BULK COLLECT, with MAX_DETAILS_TO_PUBLISH as the LIMIT clause.
- Loop through records in the BULK COLLECT table. If the record's message_type differs from the message type passed into the function, exit from the loop. Otherwise, add the data from the record to the Oracle Object being used for publication.
- Ensure that ITEMLOC_MFQUEUE is deleted from as needed.
- Ensure that ROUTING_INFO is constructed if routing information is stored at the detail level in the business transaction.

Make sure to set LP_error_status to API_CODES.UNHANDLED_ERROR before any DML statements.

A concern here is making sure that we don't delete records from the queue table that have not been published. For this reason, we do our deletes by ROWID. We also try to get everything in the same cursor. This should ensure that the message we published matches the deletes we perform from the ITEMLOC_MFQUEUE table regardless of trigger execution during GETNXT calls.

Function Level Description – BUILD_DETAIL_DELETE_OBJECTS (local)

```
Function:  BUILD_DETAIL_DELETE_OBJECTS
(O_error_msg       IN OUT    VARCHAR2,
O_ilphys_tbl       IN OUT    nocopy RIB_ITEMLOCPHYSREF_TBL,
O_routing_info     IN OUT    nocopy RIB_ROUTINGINFO_TBL,
I_message_type     IN        ITEMLOC_MFQUEUE.MESSAGE_TYPE%TYPE,
I_item             IN        ITEMLOC_MFQUEUE.ITEM%TYPE,
I_max_details      IN        rib_settings.max_details_to_publish%TYPE);
```

This function works the same way as BUILD_DETAIL_OBJECTS, except for the fact that a REF object is being created instead of a DESC object.

Function Level Description – HANDLE_ERRORS (local)

```
PROCEDURE  HANDLE_ERRORS
(O_status_code     IN OUT       VARCHAR2,
O_error_message    IN OUT       VARCHAR2,
O_message          IN OUT       nocopy RIB_OBJECT,
O_bus_obj_id       IN OUT       nocopy RIB_BUSOBJID_TBL,
O_routing_info     IN OUT       nocopy RIB_ROUTINGINFO_TBL,
I_item             IN           ITEMLOC_MFQUEUE.ITEM%TYPE,
I_physical_loc     IN           ITEMLOC_MFQUEUE.PHYSICAL_LOC%TYPE,
I_loc              IN           ITEMLOC_MFQUEUE.LOC%TYPE,
I_seq_no           IN           ITEMLOC_MFQUEUE.SEQ_NO%TYPE);
```

HANDLE_ERRORS is called from GETNXT and PUB_RETRY when an exception is raised.

If the error is a non-fatal error, GETNXT passes the sequence number of the driving ITEMLOC_MFQUEUE record back to the RIB in the ROUTING_INFO. It sends back a status of 'H'ospital to the RIB as well. It then updates the status of the queue record to 'H'ospital, so that it will not get picked up again by the driving cursor in GETNXT.

If the error is a fatal error, a status of 'E'rror is returned to the RIB.

The error is considered non-fatal if no DML has occurred yet. Whenever DML has occurred, then the global variable LP_error_status is flipped from 'H'ospital to 'E'rror.

Trigger Impact

Create a trigger on the ITEM_LOC to capture Inserts, Updates, and Deletes.

Only transaction-level items should be processed. If the item is not transaction-level, exit the trigger before calling ADDTOQ

Trigger name: EC_TABLE_ITL_AIUDR.TRG (mod)

Trigger file name: ec_table_itl_aiudr.trg (mod)

Table: ITEMLOC

Inserts

- Send the L_record (I_item, I_loc, and the I_loc_type) to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEMLOC.ITEMLOC_ADD.

Updates

- Send the L_prim_repl_supplier , L_'repl_method, L_reject_store_ord_ind,L_next_delivery_date to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEMLOC.ITEMLOC_UPD.
- The only updates that need to be captured are updates to the columns RECEIVE_AS_TYPE, SOURCE_WH, STORE_PRICE_IND, PRIMARY_SUPP, STATUS, SOURCE_METHOD, LOCAL_ITEM_DESC, PRIMARY_CNTRY, LOCAL_SHORT_DESC, and TAXABLE_IND

Deletes

- Send the L_record (I_item, I_loc, and the I_loc_type) to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEMLOC.ITEMLOC_DEL.

The trigger should fire not only for stores (loc_type = 'S') but also for warehouses (loc_type = 'W').

Trigger name: EC_TABLE_RIL_AIUDR.TRG (mod)

Trigger file name: ec_table_ril_aiudr.trg (mod)

Table: REPL_ITEM_LOC

Create a trigger on the ITEM_LOC to capture Inserts, Updates, and Deletes.

Updates

- Send the L_prim_repl_supplier , L_repl_method, L_reject_store_ord_ind,L_next_delivery_date and the L_record (I_item, I_loc, and the I_loc_type) to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEMLOC.REPL_UPD.
- The only updates that need to be captured are updates to the columns PRIMARY_REPL_SUPPLIER, REPL_METHOD, REJECT_STORE_ORD_IND, and NEXT_DELIVERY_DATE.

Deletes

- Send the L_record (I_item, I_loc, and the I_loc_type)to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEMLOC.REPL_UPD

DTD

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

Message Types	Message Type Description	Document Type Definition (DTD)
ItemLocCre	Item Loc Create Message	ItemLocDesc.dtd
ItemLocMod	Item Loc Modify Message	ItemLocDesc.dtd
ItemLocDel	Item Loc Delete Message	ItemLocRef.dtd
ItemLocReplMod	Item Loc Replenishment Modify Message	ItemLocDesc.dtd

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_MFQUEUE	Yes	No	No	No
ITEMLOC_MFQUEUE	Yes	Yes	Yes	Yes
ITEM_MASTER	Yes	No	No	No

Design Assumptions

It is not possible for a detail trigger to accurately know the status of a header table.

In order for the detail triggers to accurately know when to add a message to the queue, RMS should not allow approval of a business object while detail modifications are being made.

It is not possible for a header trigger or a detail trigger to know the status of anything modified by GETNXT. If a header trigger or detail trigger is trying to delete queue records that GETNXT currently has locked, it will have to wait until GETNXT is finished and removes the lock. It is assumed that this time will be fairly short (ie at most 2-3 seconds.) It is also assumed that this will occur rarely, as it involves updating/deleting detail records on a business object that has already been approved. This also has to occur at the same time GETNXT is processing the current business object.

Push off all DML statements as late as possible. Once DML statements have taken place, any error becomes a fatal error rather than a hospital error.

Merchandise Hierarchy Publishing API.doc

Business Overview:

RPM must know the merchandise hierarchy values that RMS contains. To ensure that RPM has the most current merchandise hierarchy values that RMS has, a new publishing API will be created to send the merchandise hierarchy information to the RIB so that RPM may subscribe to it.

Functionality Checklist:

Description	RMS	RIB
RMS must publish Merchandise Hierarchy information		
Create new Publisher	X	X

Form Impact

None.

Business Object Records

N/A

Package Impact

Business object id

The RIB uses the business object id to determine message dependencies when sending messages to a subscribing application. If a Create message has already failed in the subscribing application, and a Modify/Delete message is about to be sent from the RIB to the subscribing application, the RIB will not send the Modify/Delete message if it has the same business object id as the failed Create message. Instead, the Modify/Delete message will go directly to the hospital.

If the message relates to Districts, the business object id will be the district. If the message relates to groups, the business object id will be the group number. If the message relates to a department, the department number is the business object id. If the message relates to a class, the business object id will be the department number and the class number. Finally, if the message relates to a subclass, the business object id will be the department, class and subclass.

Package name: RMSMFM_MERCHHIER

Spec file name: rmsmfm_merchhiers.pls

Body file name: rmsmfm_merchhierb.pls

Package Specification – Global Variables

FAMILY	CONSTANT	RIB_SETTINGS.FAMILY%TYPE	:= 'merchhier';
DIV_ADD	CONSTANT	VARCHAR2(64)	:= 'divisioncre';
DIV_UPD	CONSTANT	VARCHAR2(64)	:= 'divisionmod';
DIV_DEL	CONSTANT	VARCHAR2(64)	:= 'divisiondel';
GRP_ADD	CONSTANT	VARCHAR2(64)	:= 'groupcre';
GRP_UPD	CONSTANT	VARCHAR2(64)	:= 'groupmod';
GRP_DEL	CONSTANT	VARCHAR2(64)	:= 'groupdel';
DEP_ADD	CONSTANT	VARCHAR2(64)	:= 'deptcre';
DEP_UPD	CONSTANT	VARCHAR2(64)	:= 'deptmod';
DEP_DEL	CONSTANT	VARCHAR2(64)	:= 'deptdel';
CLS_ADD	CONSTANT	VARCHAR2(64)	:= 'classcre';
CLS_UPD	CONSTANT	VARCHAR2(64)	:= 'classmod';
CLS_DEL	CONSTANT	VARCHAR2(64)	:= 'classdel';
SUB_ADD	CONSTANT	VARCHAR2(64)	:= 'subclasscre';
SUB_UPD	CONSTANT	VARCHAR2(64)	:= 'subclassmod';
SUB_DEL	CONSTANT	VARCHAR2(64)	:= 'subclassdel';

Package Body – Global Variables

```
     LP_seq_no       MERCHHIER_MFQUEUE.SEQ_NO%TYPE  := NULL;
       LP_error_status VARCHAR2(1)                    := NULL;

      cursor C_QUEUE( P_thread_val in number) is
         select q.rowid,
                q.seq_no,
                q.division,
                q.group_no,
                q.dept,
                q.class,
                q.subclass,
                q.div_name,
                q.buyer,
                q.merch,
                q.total_market_amount,
                q.group_name,
                q.dept_name,
                q.profit_calc_type,
                q.purchase_type,
                q.bud_int,
                q.bud_mkup,
                q.markup_calc_type,
                q.otb_calc_type,
                q.dept_vat_incl_ind,
                q.class_name,
                q.class_vat_ind,
                q.subclass_name,
                q.message_type,
                q.pub_status
           from merchhier_mfqueue q
          where q.seq_no = nvl(LP_seq_no,(select min(q2.seq_no)

   from merchhier_mfqueue q2
                                                                     where
   q2.thread_no = nvl(P_thread_val, q2.thread_no)

   and q2.pub_status = 'U'))
               for update NOWAIT;
```

Function Level Description – ADDTOQ

Function: ADDTOQ

(O_error_msg	OUT	VARCHAR2,
I_message_type	IN	MERCHHIER_MFQUEUE.MESSAGE_TYPE%TYPE,
I_division	IN	DIVISION.DIVISION%TYPE,
I_division_rec	IN	DIVISION%ROWTYPE,
I_group_no	IN	GROUPS.GROUP_NO%TYPE,
I_groups_rec	IN	GROUPS%ROWTYPE,
I_dept	IN	DEPS.DEPT%TYPE,
I_deps_rec	IN	DEPS%ROWTYPE,
I_class	IN	CLASS.CLASS%TYPE,
I_class_rec	IN	CLASS%ROWTYPE,
I_subclass	IN	SUBCLASS.SUBCLASS%TYPE,
I_subclass_rec	IN	SUBCLASS%ROWTYPE)

If multi-threading is being used, call API_LIBRARY.RIB_SETTINGS to get the number of threads used for the publisher. Using the number of threads, and the business object id, calculate the thread value.

Insert a record into the MERCHHIER_MFQUEUE.

Function Level Description – GETNXT

Procedure: GETNXT

(O_status_code	OUT	VARCHAR2,
O_error_msg	OUT	VARCHAR2,
O_message_type	OUT	VARCHAR2,
O_message	OUT	RIB_OBJECT,
O_bus_obj_id	OUT	RIB_BUSOBJID_TBL,
O_routing_info	OUT	RIB_ROUTINGINFO_TBL,
I_num_threads	IN	NUMBER DEFAULT 1,
I_thread_val	IN	NUMBER DEFAULT 1)

The RIB calls GETNXT to get messages. The procedure will use the C_QUEUE cursor defined in the specification of the package body to find the next message on the MERCHHIER_MFQUEUE to be published to the RIB.

After retrieving a record from the queue table, GETNXT checks for records on the queue with a status of 'H'ospital. If there are any such records for the current business object, GETNXT should raise an exception to send the current message to the Hospital.

The information from the MERCHHIER_MFQUEUE table is passed to PROCESS_QUEUE_RECORD. PROCESS_QUEUE_RECORD will build the Oracle Object message to pass back to the RIB. If PROCESS_QUEUE_RECORD does not run successfully, GETNXT should raise an exception.

After PROCESS_QUEUE_RECORD returns an oracle object to pass to the RIB, this procedure will delete the record on MERCHHIER_MFQUEUE that was just processed. If any exception is raised in GETNXT, including the exception raised by an unsuccessful call to PROCESS_QUEUE_RECORD, HANDLE_ERRORS should be called.

Function Level Description – PUB_RETRY

Procedure: PUB_RETRY

(O_status_code	OUT	VARCHAR2,
O_error_msg	OUT	VARCHAR2,
O_message_type	IN OUT	VARCHAR2,
O_message	OUT	RIB_OBJECT,
O_bus_obj_id	IN OUT	RIB_BUSOBJID_TBL,
O_routing_info	IN OUT	RIB_ROUTINGINFO_TBL,
I_REF_OBJECT	IN	RIB_OBJECT);

Same as GETNXT except:

> The record on MERCHHIER_MFQUEUE must match the passed in sequence number (contained in the ROUTING_INFO).

Function Level Description – PROCESS_QUEUE_RECORD (local)

```
Procedure: PROCESS_QUEUE_RECORD
(O_error_message     OUT       VARCHAR2,
O_message            IN  OUT   nocopy  RIB_OBJECT,
O_bus_obj_id         IN  OUT   nocopy  RIB_BUSOBJID_TBL,
I_message_type       IN        VARCHAR2,
I_rec                IN        C_QUEUE%ROWTYPE)
```

This function controls the building of Oracle Objects given the business transaction's key values and a message type. It contains all of the shared processing between GETNXT and PUB_RETRY.

If the record from MERCHHIER_MFQUEUE table is an add or update (DIV_ADD, DIV_UPD, GRP_ADD, etc…)

- Build the appropriate ref Oracle Object to publish to the RIB.

If the record from MERCHHIER _MFQUEUE table is a delete (DIV_DEL, GRP_DEL, etc…)

- Build the appropriate ref Oracle Object to publish to the RIB.

In addition to building the oracle objects, this function will populate the business object id. If the message is for a division, group or department, the business object id will be the division, group, or department respectively. If the message is for a class, the business object will be the class and department combination. If the message is for a subclass, the business object id will be the subclass, class and department combination.

Function Level Description – HANDLE_ERRORS (local)

```
PROCEDURE HANDLE_ERRORS
(O_status_code     IN  OUT    VARCHAR2,
O_error_msg        IN  OUT    VARCHAR2,
O_message          OUT        RIB_OBJECT,
O_message_type     IN  OUT    VARCHAR2,
O_bus_obj_id       IN  OUT    RIB_BUSOBJID_TBL,
O_routing_info     IN  OUT    RIB_ROUTINGINFO_TBL,
I_rec              IN         C_QUEUE%TYPE)
```

HANDLE_ERRORS is called from GETNXT and PUB_RETRY when an exception is raised.

If the error is a non-fatal error, GETNXT passes the sequence number of the driving MERCHHIER_MFQUEUE record back to the RIB in the ROUTING_INFO. It sends back a status of 'H'ospital to the RIB as well. It then updates the status of the queue record to 'H'ospital, so that it will not get picked up again by the driving cursor in GETNXT.

If the error is a fatal error, a status of 'E'rror is returned to the RIB.

The error is considered non-fatal if no DML has occurred yet. Whenever DML has occurred, then the global variable LP_error_status is flipped from 'H'ospital to 'E'rror.

Trigger Impact

Trigger name: EC_TABLE_DIV_AIUDR.TRG

Trigger file name: ec_table_div_aiudr.trg

Table: DIVISION

Create a trigger on the DIVISION table to capture Inserts, Updates, and Deletes.

Inserts

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.DIV_ADD.

Updates

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.DIV_UPD.

Deletes

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.DIV_DEL.

Trigger name: EC_TABLE_GRO_AIUDR.TRG

Trigger file name: ec_table_gro_aiudr.trg

Table: GROUPS

Create a trigger on the GROUPS table to capture Inserts, Updates, and Deletes.

Inserts

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.GRP_ADD.

Updates

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.GRP_UPD.

Deletes

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.GRP_DEL.

Trigger name: EC_TABLE_DEP_AIUDR.TRG

Trigger file name: ec_table_dep_aiudr.trg

Table: DEPS

Create a trigger on the DEPS table to capture Inserts, Updates, and Deletes.

Inserts

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.DEP_ADD.

Updates

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.DEP_UPD.

Deletes

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.DEP_DEL.

Trigger name: EC_TABLE_CLA_AIUDR.TRG

Trigger file name: ec_table_cla_aiudr.trg

Table: CLASS

Create a trigger on the CLASS table to capture Inserts, Updates, and Deletes.

Inserts

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.CLS_ADD.

Updates

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.CLS_UPD.

Deletes

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.CLS_DEL.

Trigger name: EC_TABLE_SCL_AIUDR.TRG

Trigger file name: ec_table_scl_aiudr.trg

Table: SUBCLASS

Create a trigger on the SUBCLASS table to capture Inserts, Updates, and Deletes.

Inserts

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.SUB_ADD.

Updates

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.SUB_UPD.

Deletes

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.SUB_DEL.

DTD

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

Message Types	Message Type Description	Document Type Definition (DTD)
divisoncre	Division Create Message	MrchHrDivDesc.dtd
divisonmod	Division Modify Message	MrchHrDivDesc.dtd
divisiondel	Division Delete Message	MrchHrDivRef.dtd
groupcre	Group Detail Create Message	MrchHrGrpDesc.dtd
groupmod	Group Detail Modify Message	MrchHrGrpDesc.dtd
groupdel	Group Detail Delete Message	MrchHrGrpRef.dtd
deptcre	Department Detail Create Message	MrchHrDeptDesc.dtd
deptmod	Department Detail Modify Message	MrchHrDeptDesc.dtd
deptdel	Department Detail Delete Message	MrchHrDeptRef.dtd
classcre	Class Detail Create Message	MrchHrClsDesc.dtd
classmod	Class Detail Modify Message	MrchHrClsDesc.dtd
classdel	Class Detail Delete Message	MrchHrClsRef.dtd
subclasscre	Subclass Detail Create Message	MrchHrSclsDesc.dtd
subclassmod	Subclass Detail Modify Message	MrchHrSclstDesc.dtd
subclassdel	Subclass Detail Delete Message	MrchHrSclsRef.dtd

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
MERCHHIER_MFQUEUE	Yes	Yes	Yes	Yes
DIVISION	Yes	No	No	No
DEPT	Yes	No	No	No
CLASS	Yes	No	No	No
SUBCLASS	Yes	No	No	No

Assumptions

Push off all DML statements as late as possible. Once DML statements have taken place, any error becomes a fatal error rather than a hospital error.

Regular Price Change Subscription API.doc

Functional Area

RMS subscribing to regular price changes

Business Overview

RPM will now send approved Price Changes for Regular price items to SIM. However, RMS also needs to know this information so it can set the effective date of the price change on the TICKET_REQUEST table.

A new subscription family will be created to bring price changes from RPM to RMS. The new sub family will be called RegPrcChg. RMS will subscribe the create, mod and delete of price change messages.

The new subscription API will call the new functions within TICKET_SQL package to create, modify or delete the price change record on the TICKET_REQUEST table. Packs and Warehouse locations will be ignored.

Once the price changes have already been downloaded by tckdnld.pc, there could be instances where a new price change is updated or deleted for the same price change ID/Item/Loc/Date combination. In this case, the latest instance of price change will be inserted or updated in TICKET_REQUEST table.

Package Impact

Filename: rmssub_regprcchgtkts/b.pls

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

This procedure initially checks that the passed in message type is a valid type for Regular Price Change messages. The valid message types are: regprcchgCre, regprcchgMod and regprcchgDel. If the message type is invalid, a status of “E” should be returned to the external system along with an appropriate error message informing the external system that the status is invalid.

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

If the downcast is successful, then consume needs to verify that the message passes all of RMS's business validation. It does not actually perform any validation itself; instead, it calls the RMSSUB_REGPRCCHG_TKT_VALIDATE.CHECK_MESSAGE function to determine whether the message is valid. This function is overloaded so simply passing the object in should be sufficient. If the message passed RMS business validation, then the function will return true, otherwise it will return false. If the message has failed RMS business validation, a status of "E" should be returned to the external system along with the error message returned from the CHECK_MESSAGE function.

Once the message has passed RMS business validation, it can be persisted to the RMS database. The consume function does not have to have any knowledge of how to persist the message to the database, it calls the RMSSUB_REGPRCCHG_TKT_SQL.PERSIST() function. This function is overloaded so simply passing the object should be sufficient. If the database persistence fails, the function will return false. A status of "E" should be returned to the external system along with the error message returned from the PERSIST() function.

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

RMSSUB_REGPRCCHG_TKT.HANDLE_ERROR() – This is the standard error handling function that wraps the API_LIBRARY.HANDLE_ERROR function.

Business Validation Module

File name: rmssub_regprcchg_tktvals.pls

Global Function: CHECK_MESSAGE

```
(O_error_mesage     OUT     RTK_ERRORS.RTK_TEXT%TYPE,
O_ticket_tbl        OUT     TICKET_SQL.TICKET_TBL,
I_message           IN      RIB_REGPRCCHGDESC_REC,
I_message_type      IN      VARCHAR2)
```

This overloaded function performs all business validation associated with ticket request create, modify and delete messages. It is important that the signature uses IN for the message and not IN OUT. When IN is used, the parameter is passed by reference. Passing by reference keeps the server from duplicating the memory allocation.

Call the CHECK_REQUIRED_FIELDS function to make sure all required fields are not NULL. Call the CHECK_EXISTENCE function to see if a record for the item/loc/price change ID already exists on TICKET_REQUEST table. If it exists and the message type is Create, treat it as a Mod message type. If it does NOT exists and the message type is Mod, treat it as a Create message type. If it does not exist and the message type is Delete, return TRUE since no further processing is needed. Call VALIDATE_MESSAGE function to check for validity of the required fields.

Finally, the ticket record used for DML is populated within the POPULATE_RECORD function and passed back to RMSSUB_REGPRCCHG_TKT.CONSUME.

Internal Functions:

```
Function: CHECK_REQUIRED_FIELDS
(O_error_message        OUT         RTK_ERRORS.RTK_TEXT%TYPE,
I_message               IN          RIB_REGPRCCHGDESC_REC)
```

This overloaded function ensures that all required fields in the message are NOT NULL.

```
VALIDATE_MESSAGE
(O_error_message        IN OUT          RTK_ERRORS.RTK_TEXT%TYPE,
I_message               IN              RIB_REGPRCCHGDESC_REC)
```

This overloaded function validates the values of the message. It will also populate the fields on TICKET_REQUEST that were not included in the message.

- If the location type is warehouse or the item is a Pack, no further processing is needed.
- Stores are validated along with the currency code.
- Validate that the item is at the transactional level.

```
POPULATE_RECORD
(O_error_message            OUT         RTK_ERRORS.RTK_TEXT%TYPE,
O_ins_ticket_tbl            OUT         TICKET_SQL.TICKET_TBL,
O_upd_ticket_tbl            OUT         TICKET_SQL.TICKET_TBL,
I_message                   IN          RIB_REGPRCCHGDESC_REC)
```

This overloaded function populates the ticket request output record with the values sent in the message.

Bulk or single DML module

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

File name: rmssub_regprcchg_tktsqls.pls

```
PERSIST
(O_error_mesage         OUT         RTK_ERRORS.RTK_TEXT%TYPE,
I_message               IN          TICKET_SQL.TICKET_TBL,
I_upd_message           IN          TICKET_SQL.TICKET_TBL,
I_message_type          IN          VARCHAR2)
```

This function will perform INSERT/UPDATE/DELETE statements by calling the appropriate functions according to the message type and passing the data in a record to these functions.

For the message type indicating a ticket request insert or update, call the TICKET_SQL.NEW_PRICE_CHANGE and TICKET_SQL.UPDATE_PRICE_CHANGE functions with the ticket record. For the message type indicating a delete, call TICKET_SQL.DELETE_PRICE_CHANGE function to delete the record from TICKET_REQUEST.

File name: tickets/b.pls

Add the following new functions to the existing package.

```
NEW_PRICE_CHANGE
(O_error_message         OUT     RTK_ERRORS.RTK_TEXT%TYPE,

I_ticket_tbl             IN      TICKET_SQL.TICKET_TBL)
```

This function inserts records into the TICKET_REQUEST table.

```
UPDATE_PRICE_CHANGE
(O_error_message         OUT          RTK_ERRORS.RTK_TEXT%TYPE,
I_ticket_tbl             IN           TICKET_SQL.TICKET_TBL)
```

This function calls LOCK_TICKET function to lock and update the TICKET_REQUEST table.

```
LOCK_TICKET
(O_error_message              OUT          RTK_ERRORS.RTK_TEXT%TYPE,
I_ticket_tbl                  IN           TICKET_SQL. TICKET_TBL)
```

This function locks the records on TICKET_REQUEST table.

```
DELETE_PRICE_CHANGE
(O_error_message              OUT          RTK_ERRORS.RTK_TEXT%TYPE,
I_ticket_tbl                  IN           TICKET_SQL.TICKET_TBL)
```

This function calls LOCK_TICKET function to lock and delete records from the TICKET_REQUEST table.

Message DTD

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

Message Types	Message Type Description	Document Type Definition (DTD)
RegPrcChgCre	Regular Price Change Create Message	RegPrcChgDesc.dtd
RegPrcChgMod	Regular Price Change Modify Message	RegPrcChgDesc.dtd
RegPrcChgDel	Regular Price Change Delete Message	RegPrcChgDesc.dtd

Design Assumptions

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

Tables

TABLE	SELECT	INSERT	UPDATE	DELETE
TICKET_REQUEST	Yes	Yes	Yes	Yes
ITEM_LOC_SOH	Yes	No	No	No
STORE	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
ITEM_TICKET	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No

Transfers Publication API.doc

Business Overview:

Transfers consist of header level information in which source and destination locations are specified, and detail information regarding what items and how much of each item is to be transferred. Both of the transfer tables, tsfhead and tsfdetail, will now have triggers that track inserts, deletes, and modifications. These triggers will insert or update into TSF_MFQUEUE or TRANSFERS_PUB_INFO tables. The transfer family manager will be responsible for pulling transfer information from this queue and sending it to the external system(s) at the appropriate time and in the correct order.

The transfer messages that will be published by the family manager will vary. A complete message including header information, detail information, and component ticketing information (if applicable) will be created when a transfer is approved. . When the transfer is unapproved, the RIB treats it like a TransferDel message when publishing it to external systems. When the transfer is re-approved, the transfer is treated like a new transfer for publishing.

RMS 11 Transfers include a Context information, at the header level. The context_type will define the business reason for the Transfer, thus allowing users to distinguish one form of Transfer from another. The context_value will further identify a specific context_type. For example, when the Context of a Transfer is Promotion (i.e. the Transfer is being created to support an RPM Promotion), the ID of the Promotion being supported will be attached to the Transfer as well, at the header level.

Functionality Checklist:

Description	RMS	RIB
RMS must publish Transfers information		
Create new Publisher	X	X

Form Impact

None.

Business Object Records

Create the following table types in the RMSMFM_TRANSFERs package:

```
TYPE rowid_TBL      is table of ROWID INDEX BY BINARY_INTEGER;
```

Package Impact

Create Header

1 Prerequisites: None.

2 Activity Detail: The first step to creating a transfer is creating the header level information.

3 Messages: When a transfer is created, a record is inserted into TRANSERS_PUB_INFO table and is not published onto the queue until the transfer has been approved.
Approve

Approve

1 Prerequisites: A transfer must exist and have at least one detail before it can be approved.

2 Activity Detail: Approving a transfer changes the status of the transfer. This change in status signifies the first time systems external to RMS will have an interest in the existence of the transfer, so this is the first part of the life cycle of a transfer that is published.

3 Messages: When a transfer is approved, a “TransferHdrMod” message is inserted into the queue with the appr_ind on the queue set to ‘Y’ signifying that the transfer was approved. The family manager uses this indicator to create a hierarchical message containing a full snapshot of the transfer at the time the message is published.

Modify Header

1 Prerequisites: The transfer header can only be modified when the status is NOT approved. Once the transfer is approved, the only fields that are modifiable are the status field and the comments field.

2 Activity Detail: The user is allowed to modify the header but only certain fields at certain times. If a transfer is in input status the to and from locations may be modified until details have been added. Once details have been added, the locations are disabled. The freight code is modifiable until the transfer has been approved. Comments can be modified at any time.

3 Messages: When the status of the header is either changed to ‘C’losed or ‘A’pproved, a message (TransferHdrMod) is inserted into the queue. (Look above at Approve activity and below at Close activity for further details).

Create Details

1 Prerequisites: A transfer header record must exist before transfer details can be created.

2 Activity Detail: The user is allowed to add items to a transfer but only until it has been approved. Once a transfer has been approved, details can longer be added.

3 Messages: No messages are created on the queue until the transfer is approved.

Modify Details

1 Prerequisites: Only modifications to transfer quantities will be sent to the queue, and only when the transfer quantity is decreased manually, and not because of an increase in cancelled quantity will it be sent to the queue.

2 Activity Detail: The user is allowed to change transfer quantities provided they are not reduced below those already shipped. The transfer quantity can also be decreased by an increase in the cancelled quantity - which is always initiated by the external system. This change, then, would be of no interest to the external system because it was driven by it.

3 Messages: No messages are created on the queue until the transfer is approved.

Delete Details

1 Prerequisites: Only a detail that hasn’t been shipped may be deleted and it cannot be deleted if it is currently being worked on by an external system. A user is not allowed to delete details from a closed transfer.

2 Activity Detail: A user is allowed to delete details from a transfer but only if the item hasn’t been shipped.

3 Messages: No messages are created on the queue until the transfer is approved.

Close

1 Prerequisites: A transfer must be in shipped status before it can be closed, and it cannot be in the process of being worked on by an external system.

2 Activity Detail: Closing a transfer changes the status, which prevents any further modifications to the transfer. When a transfer is closed, a message is published to update the external system(s) that the transfer has been closed and no further work (in RMS) will be performed on it.

3 Messages: Closing a transfer queues a “TransferHdrMod” request. This is a flat message containing a snapshot of the transfer header information at the time the message is published.

Delete

1 Prerequisites: A transfer can only be deleted when it is still in approved status or when it has been closed.

2 Activity Detail: Deleting a transfer removes it from the system. External systems are notified by a published Delete message that contains the number of the transfer to be deleted.

3 Message: When a transfer is deleted, a “TransferDel”, which is a flat notification message, is queued.

Package name: RMSMFM_TRANSFERS

Spec file name: rmsmfm_transferss.pls

Body file name: rmsmfm_transfersb.pls

Package Specification – Global Variables

```
FAMILY      VARCHAR2(64) := 'transfers';


HDR_ADD     VARCHAR2(64) := 'TransferCre';
HDR_UPD     VARCHAR2(64) := 'TransferHdrMod';
HDR_DEL     VARCHAR2(64) := 'TransferDel';
HDR_UNAPRV  VARCHAR2(64) := 'TransferUnapp';
DTL_ADD     VARCHAR2(64) := 'TransferDtlCre';
DTL_UPD     VARCHAR2(64) := 'TransferDtlMod';
DTL_DEL     VARCHAR2(64) := 'TransferDtlDel';
```

Function Level Description – ADDTOQ

```
FUNCTION ADDTOQ
(O_error_mesage    OUT   VARCHAR2,
I_message_type     IN    VARCHAR2,
I_tsf_no           IN    tsfhead.tsf_no%TYPE,
I_tsf_type         IN    tsfhead.tsf_type%TYPE,
I_tsf_head_status  IN    tsfdetail.status%TYPE,
I_item             IN    tsfdetail.item%TYPE,
I_publish_ind      IN    tsfdetail.publish_ind%TYPE)
```

This function is called by both the tsfhead trigger and the tsfdetail trigger, ec_table_thd_aiudr and ec_table_tdt_aiudr respectively.

- Book Transfers, Non-Sellable Transfers and Externally Generated Transfers (except for delete messages) are never published to external systems.
- For header level insert messages (HDR_ADD), insert a record in the TRANSFERS_PUB_INFO table. The published flag should be set to ‘N’. The correct thread for the Business transaction should be calculated and written. Call API_LIBRARY.RIB_SETTINGS to get the number of threads used for the publisher. Using the number of threads, and the Business object id, calculate the thread value.
- For all records except header level inserts (HDR_ADD), the thread_no and initial_approval_ind should be queried from the TRANSFERS_PUB_INFO table.
- If the Business transaction has not been approved (initial_approval_ind = ‘N’) and the triggering message is one of DTL_ADD, DTL_UPD, DTL_DEL, HDR_DEL, no processing should take place and the function should exit.
- For detail level message deletes (DTL_DEL), we only need one (the most recent) record per detail in the TSF_MFQUEUE. Delete any previous records that exist on the TSF_MFQUEUE for the record that has been passed. If the publish_ind is ‘N’, do not add the DTL_DEL message to the queue.

- For detail level message updates (DTL_UPD), we only need one DTL_UPD (the most recent) record per detail in the TSF_MFQUEUE. Delete any previous DTL_UPD records that exist on the TSF_MFQUEUE for the record that has been passed. We don't want to delete any detail inserts that exist on the queue for the detail, we need to ensure subscribers have not passed a detail modification message for a detail that they do not yet have.
- For header level delete messages (HDR_DEL), delete every record in the queue for the Business transaction.
- For header level update message (HDR_UPD), update the TRANSFERS_PUB_INFO.INITIAL_APPROVAL_IND to 'Y' if the Business transaction is in approved status.
- For all records except header level inserts (HDR_ADD), insert a record into the TSF_MFQUEUE.

It returns a status code of API_CODES.SUCCESS if successful, API_CODES.UNHANDLED_ERROR if not.

Function Level Description – GETNXT

```
PROCEDURE GETNXT
(O_status_code       OUT      VARCHAR2,
O_error_msg          OUT      VARCHAR2,
O_message_type       OUT      VARCHAR2,
O_message            OUT      RIB_OBJECT,
O_bus_obj_id         OUT      RIB_BUSOBJID_TBL,
O_routing_info       OUT      RIB_ROUTINGINFO_TBL,
I_num_threads        IN       NUMBER DEFAULT 1,
I_thread_val         IN       NUMBER DEFAULT 1)
```

The RIB calls GETNXT to get messages. It performs a cursor loop on the unpublished records on the TSF_MFQUEUE table (PUB_STATUS = 'U'). It should only need to execute one loop iteration in most cases. For each record retrieved, GETNXT gets the following:

1 A lock of the queue table for the current Business object. The lock is obtained by calling the function LOCK_THE_BLOCK. If there are any records on the queue for the current Business object that are already locked, the current message is skipped.

2 The published indicator from the TRANSFERS_PUB_INFO table.

3 A check for records on the queue with a status of 'H'ospital. If there are any such records for the current Business object, GETNXT raises an exception to send the current message to the Hospital.

The loop will need to execute more than one iteration for the following cases:

1 When a header delete message exists on the queue for a Business object that has not been initially published. In this case, simply remove the header delete message from the queue and loop again.

2 A detail delete message exists on the queue for a detail record that has not been initially published. In this case, simply remove the detail delete message from the queue and loop again.

3 The queue is locked for the current Business object

The information from the TSF_MFQUEUE and TRANSFERS_PUB_INFO table is passed to PROCESS_QUEUE_RECORD. PROCESS_QUEUE_RECORD will build the Oracle Object message to pass back to the RIB. If PROCESS_QUEUE_RECORD does not run successfully, GETNXT raises an exception.

If any exception is raised in GETNXT, including the exception raised by an unsuccessful call to PROCESS_QUEUE_RECORD, HANDLE_ERRORS is called.

Function Level Description – PUB_RETRY

```
PROCEDURE PUB_RETRY
(O_status_code      OUT        VARCHAR2,
O_error_msg         OUT        VARCHAR2,
O_message_type      IN  OUT    VARCHAR2,
O_message           OUT        RIB_OBJECT,
O_bus_obj_id        IN  OUT    RIB_BUSOBJID_TBL,
O_routing_info      IN  OUT    RIB_ROUTINGINFO_TBL,
I_REF_OBJECT        IN         RIB_OBJECT);
```

Same as GETNXT except:

It only loops for a specific row in the TSF_MFQUEUE table. The record on TSF_MFQUEUE must match the passed in sequence number (contained in the ROUTING_INFO).

Function Level Description – PROCESS_QUEUE_RECORD (local)

```
FUNCTION PROCESS_QUEUE_RECORD
(O_error_msg        OUT        VARCHAR2,
O_break_loop        OUT        BOOLEAN,
O_message           IN  OUT    nocopy RIB_OBJECT,
O_routing_info      IN  OUT    nocopy RIB_ROUTINGINFO_TBL,
O_bus_obj_id        IN  OUT    nocopy RIB_BUSOBJID_TBL,
O_message_type      IN  OUT    VARCHAR2,
I_tsf_no            IN         tsf_mfqueue.tsf_no%TYPE,
I_hdr_published     IN         transfers_pub_info.published%TYPE,
I_item              IN         tsf_mfqueue.item%TYPE,
I_pub_status        IN         tsf_mfqueue.pub_status%TYPE,
I_seq_no            IN         tsf_mfqueue.seq_no%TYPE,
I_rowid             IN         ROWID,
O_keep_queue        IN OUT     BOOLEAN)
```

This function controls the building of Oracle Objects given the business transaction's key values and a message type. It contains all of the shared processing between GETNXT and PUB_RETRY.

If the message type is HDR_DEL or HDR_UNAPRV and it has not been published

- Call DELETE_QUEUE_REC to delete the record from TSF_MFQUEUE.

If the message type is HDR_DEL and the record has been published

- Generate a "flat" file to be sent to the RIB. Delete from TRANSFER_PUB_INFO and call DELETE_QUEUE_REC to delete from the queue.

If the message type is HDR_UNAPRV

- Treat it just like a hdr_del except the published indicator on TRANSFRERS_PUB_INFO is set to 'N'.

If the message type is HDR_ADD or DTL_ADD

- Call MAKE_CREATE to publish the entire transfer.

If the record from TSF_MFQUEUE table is HDR_UPD

- Call BUILD_HEADER_OBJECT to build the Oracle Object to publish to the RIB and delete from the queue.

If the record from TSF_MFQUEUE table is DTL_ADD or DTL_UPD

- Call BUILD_HEADER_OBJECT and BUILD DETAIL_CHANGE_OBJECTS to build the Oracle Object to publish to the RIB.

If the record from TSF_MFQUEUE table is a detail delete (DTL_DEL)

- Call BUILD HEADER_OBJECT and BUILD_DETAIL_DELETE_OBJECTS to build the Oracle Object to publish to the RIB.

Function Level Description – MAKE_CREATE (local)

```
FUNCTION MAKE_CREATE
(O_error_msg       IN  OUT     VARCHAR2,
O_message          IN  OUT     nocopy RIB_OBJECT,
O_routing_info     IN  OUT     nocopy RIB_ROUTINGINFO_TBL,
I_tsf_no           IN          tsfhead.tsf_no%TYPE,
I_seq_no           IN          tsf_mfqueue.seq_no%TYPE,
I_item             IN          item_loc.item%TYPE,
I_max_details      IN          rib_settings.max_details_to_publish%TYPE,
I_rowid            IN          ROWID,
O_keep_queue       IN  OUT     BOOLEAN)
```

This function is used to create the Oracle Object for the initial publication of a Business transaction. It combines the current message and all previous messages with the same key in the queue table to create the complete hierarchical message. It first creates a new message with the hierarchical document type. It then gets the header create message and adds it to the new message. 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 with a sequence number less than or equal to the current records sequence number. This new message is passed back to the bus. The MAKE_CREATE function will not be called unless the appr_ind on the queue is 'Y'es (meaning the transfer has been approved, and it's ready to be published for the first time to the external system(s)).

Function Level Description – BUILD_HEADER_OBJECT (local)

```
FUNCTION BUILD_HEADER_OBJECT
(O_error_msg        IN  OUT     VARCHAR2,
O_message           IN  OUT     nocopy RIB_OBJECT,
O_rib_tsfref_rec    IN  OUT     nocopy RIB_TSFREF_REC,
O_routing_info      IN  OUT     nocopy RIB_ROUTINGINFO_TBL,
O_freight_code      IN  OUT     tsfhead.freight_code%TYPE,
O_to_loc_type       IN   OUT    item_loc.loc_type%TYPE,
O_from_loc_type   IN  OUT       item_loc.loc_type%TYPE,
I_tsf_no            IN  OUT     tsfhead.tsf_no%TYPE)
```

Accepts header key values, performs necessary lookups, builds and returns a header level Oracle Object.

Function Level Description – BUILD_DETAIL_OBJECTS (local)

```
FUNCTION BUILD DETAIL_OBJECTS
(O_error_msg              IN  OUT     VARCHAR2,
O_message                 IN  OUT     nocopy RIB_TSFDTL_TBL,
O_tsf_mfqueue_rowid       IN OUT      nocopy rowid_TBL,
O_tsf_mfqueue_size        IN OUT      BINARY_INTEGER,
O_tsfdetail_rowid         IN OUT      nocopy rowid_TBL,
O_tsfdetail_size          IN OUT      BINARY_INTEGER,
O_delete_rowid_ind        IN OUT      VARCHAR2,
I_message_type            IN OUT      tsf_mfqueue.message_type%TYPE,
I_tsf_no                  IN          tsfhead.tsf_no%TYPE,
I_to_loc                  IN          item_loc.loc%TYPE,
I_to_loc_type             IN          item_loc.loc_type%TYPE,
I_max_details             IN          rib_settings.max_details_to_publish%TYPE,
I_freight_code            IN          VARCHAR2)
```

This function is responsible for fetching the detail info and ticket type to be sent to RDM. The logic that gets the detail info as well as the ticket type was separated to remove the primary key constraint.

Function Level Description – BUILD_SINGLE_DETAIL (local)

```
FUNCTION BUILD_SINGLE_DETAIL
(O_error_msg        IN  OUT    VARCHAR2,
O_message           IN  OUT    nocopy RIB_TSFDTL_TBL,
IO_rib_tsfdtl_rec   IN  OUT    nocopy RIB_TSFDTL_REC,
I_tsf_no            IN         tsfhead.tsf_no%TYPE,
I_item              IN         item_master.item%TYPE,
I_pack_ind          IN         item_master.pack_ind%TYPE,
I_sellable_ind      IN         item_master.sellable_ind%TYPE,
I_store_ord_mult    IN         item_loc.store_ord_mult%TYPE,
I_tsf_po_link_no    IN         tsfdetail.tsf_po_link_no%TYPE,
I_ticket_type_id    IN         item_ticket.ticket_type_id%TYPE,
I_tsf_qty           IN         tsfdetail.tsf_qty%TYPE,
I_freight_code      IN         tsfhead.freight_code%TYPE,
I_to_loc            IN         item_loc.loc%TYPE,
I_to_loc_type       IN         item_loc.loc_type%TYPE,
I_inv_status        IN         inv_status_types.inv_status%TYPE,
I_message_type      IN         tsf_mfqueue.message_type%TYPE)
```

Accept inputs and build a detail level Oracle Object. Perform any lookups needed to complete the Oracle Object.

Function Level Description – GET_RETAIL(local)

```
FUNCTION GET_RETAIL
(O_error_msg        IN  OUT      VARCHAR2,
I_item              IN           item_loc.item%TYPE,
I_loc               IN           item_loc.loc%TYPE,
I_loc_type          IN           item_loc.loc_type%TYPE,
O_price             IN  OUT      item_zone_price.unit_retail%TYPE,
O_selling_uom       IN  OUT      item_zone_price.selling_uom%TYPE)
```

Calls PRICING_ATTRIB_SQL.GET_RETAIL to get the price and selling uom of the item.

Function Level Description – GET_GLOBALS(local)

```
FUNCTION GET_GLOBALS
(O_error_msg              IN OUT        VARCHAR2,
O_days                    OUT           NUMBER,
O_default_order_type      OUT           system_options.default_order_type%TYPE)
```

Get all the system options and variables needed for processing.

Function Level Description – GET_TSF_ENTITIES(local)

```
FUNCTION GET_TSF_ENTITIES
(O_error_msg            IN OUT          VARCHAR2,
O_from_tsf_entity       IN OUT          tsf_entity.tsf_entity_id%TYPE,
O_to_tsf_entity         IN  OUT         tsf_entity.tsf_entity_id%TYPE,
I_from_loc              IN              item_loc.loc%TYPE,
I_from_loc_type         IN              item_loc.loc_type%TYPE,
I_to_loc                IN              item_loc.loc%TYPE,
I_to_loc_type           IN              item_loc.loc_type%TYPE)
```

Get the to and from location entities for the transfer.

Function Level Description – BUILD_DETAIL_CHANGE_OBJECTS (local)

```
FUNCTION BUILD_DETAIL_CHANGE_OBJECTS
(O_error_msg            IN  OUT         VARCHAR2,
O_rib_tsfdesc_rec       IN  OUT         nocopy RIB_TSFDESC_REC,
I_message_type          IN  OUT         tsf_mfqueue.message_type%TYPE,
I_tsf_no                IN              tsfhead.tsf_no%TYPE,
I_item                  IN              item_loc.item%TYPE,
I_freight_code          IN              tsfhead.freight_cod%TYPE,
I_max_details           IN              rib_settings.max_details_to_publish%TYPE)
```

Call BUILD_DETAIL_OBJECT to publish the record. Update TSFDETAIL.publish_ind to ‘Y’ and delete the record from TSF_MFQUEUE.

Function Level Description – BUILD_DETAIL_DELETE_OBJECTS (local)

```
FUNCTION BUILD_DETAIL_DELETE_OBJECTS
(O_error_msg      IN  OUT  VARCHAR2,
O_rib_tsfref_rec  IN  OUT  nocopy RIB_TSFREF_REC,
I_tsf_no            IN          tsfhead.tsf_no%TYPE,
I_max_details     IN          rib_settings.max_details_to__ publish%TYPE)
```

Either pass in a header level ref Oracle Object or build a header level ref Oracle Object.

Perform a cursor for loop on TSF_MFQUEUE and build as many detail ref Oracle Objects as possible without exceeding the MAX_DETAILS_TO_PUBLISH.

Delete from TSF_MFQUEUE when done.

Function Level Description – LOCK_THE_BLOCK (local)

```
FUNCTION LOCK_THE_BLOCK
(O_error_msg        OUT         VARCHAR2,
O_queue_locked      OUT         BOOLEAN,
I_tsf_no            IN          tsf_mfqueue.tsf_no%TYPE)
```

This function locks all queue records for the current business object. This is to ensure that GETNXT does not wait on any business processes that currently have the queue table locked and have not committed. This can occur because ADDTOQ, which is called from the triggers, deletes from from the queue table for DTL_UPD, DTL_DEL, and HDR_DEL messages.

Function Level Description – LOCK_THE_BLOCK (local)

```
FUNCTION LOCK_DETAILS
(O_error_msg        OUT         VARCHAR2,
I_tsf_no            IN          tsf_head.tsf_no%TYPE)
```

Lock the transfer details before updating the publish_ind on TSFDETAIL.

Function Level Description – DELETE_QUEUE_REC (local)

```
FUNCTION DELETE_QUEUE_REC
(O_error_msg        OUT         VARCHAR2,
I_seq_no            IN          tsf_mfqueue.seq_no%TYPE)
```

This procedure deletes a specific record from TSF_MFQUEUE. It deletes based on the sequence number passed in.

Function Level Description – HANDLE_ERRORS (local)

```
PROCEDURE HANDLE_ERRORS
(O_status_code         IN  OUT       VARCHAR2,
O_error_msg            IN  OUT       VARCHAR2,
O_message              OUT           RIB_OBJECT,
O_message_type         IN  OUT       VARCHAR2,
O_bus_obj_id           IN  OUT       RIB_BUSOBJID_TBL,
O_routing_info         IN  OUT       RIB_ROUTINGINFO_TBL,
I_tsf_no               IN            tsf_mfqueue.tsf_no%TYPE,
I_seq_no               IN            tsf_mfqueue.seq%TYPE,
I_item                 IN            tsf_mfqueue.item%TYPE)
```

HANDLE_ERRORS is called from GETNXT and PUB_RETRY when an exception is raised. The function was updated to conform with the changes made to the ADDTOQ function.

Trigger Impact

Create a trigger on the TSFHEAD and TSFDETAIL to capture Inserts, Updates, and Deletes.

Trigger name: EC_TABLE_THD_AIUDR.TRG

Trigger file name: ec_table_thd_aiudr.trg

Table: TSFHEAD

Inserts

- Send the tsf_no and tsf_type level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_Transfers.HDR_ADD.

Updates

- Send the tsf_no and tsf_type level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_Transfers.HDR_UPD.

Deletes

- Send the tsf_no and tsf_type level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_Transfers.HDR_DEL.

Trigger name: EC_TABLE_TDT_AIUDR.TRG

Trigger file name: ec_table_tdt_aiudr.trg

Table: TSFDETAIL

Inserts

- Send the tsf_no and item level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_Transfers.DTL_ADD

Updates

- Send the tsf_no and item level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_Transfers.DTL_UPD.

Deletes

- Send the tsf_no and item level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_Transfers.DTL_DEL.

DTD

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

Message Types	Message Type Description	Document Type Definition (DTD)
TransferCre	Transfer Create Message	TransferDesc.dtd
TransferHdrMod	Transfer Modify Message	TransferDesc.dtd
TransferDel	Transfer Delete Message	TransferRef.dtd
TransferDtlCre	Transfer Detail Create Message	TransferDtlDesc.dtd
TransferDtlMod	Transfer Detail Modify Message	TransferDtlDesc.dtd
TransferDtlDel	Transfer Detail Delete Message	TransferDtlRef.dtd

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
TRANSFERS_PUB_INFO	Yes	Yes	Yes	Yes
TSF_MFQUEUE	Yes	Yes	Yes	Yes
TSF_DETAIL	Yes	No	Yes	No
TSF_HEAD	Yes	No	No	No
WH	Yes	No	No	No
ORDCUST	Yes	No	No	No
CUSTOMER	Yes	No	No	No

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_LOC	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
ITEM_TICKET	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
CODE_DETAIL	Yes	No	No	No
SYSTEM_OPTIIONS	Yes	No	No	No
RIB_SETTINGS	Yes	No	No	No

Design Assumptions

- After a transfer has been approved we are assuming the freight code of the transfer (on the tsfhead table) cannot be updated.
- 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.

Chapter 2 – Batch designs

Correction to POSUPLD.PC design

The description for the program POSUPLD.PC is inaccurate in the RMS 11.0 Operations Guide. The RMS 11.0 Operations Guide incorrectly identifies the Vdate as the weekly PO/invoice generation setting. The correct value is the ‘end of week’ date for the weekly/PO invoice generation setting.

Deal Actuals [dealact]

Design Overview

For complex deals, this new batch will run on a daily basis and retrieve data from various sources to calculate the actuals information to update the DEAL_ACTUALS_ITEM_LOC table.

Processing of actuals

The deal revenue batch runs daily to process active bill back deals. Using temporary tables created by prepost.pc, records are Inserted (if new)/updated (accumulated - if it exists) to the DEAL_ACTUALS_ITEM_LOC table at the deal, deal component, item location level for reporting periods.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DEAL_ACTUALS_FORCAST	No	Yes	No	No	No
DEAL_BB_NO_REBATE_TEMP	No	Yes	No	No	No
DEAL_BB_REBATE_PO_TEMP	No	Yes	No	No	No
DEAL_TRAN_DATA_TEMP	No	Yes	No	No	No
DEAL_HEAD	No	Yes	No	No	No
DEAL_ACTUALS_ITEM_LOC	No	Yes	Yes	Yes	No

Stored Procedures / Shared Modules (Maintainability)

Batch library module: currconv.pc used for currency conversions

DB packaged function to convert Qtys between UOMs – UOM_SQL.CONVERT.

Function Level Description

main()

This function will Validate program arguments and logon to Oracle, call the init() function to initialize restart / recovery and variables, call the process() to execute main program logic and then call the final() to clean up all internal processing.

init()

This function calls the standard initialization function retek_init() to initialize restart / recovery.

It then retrieves system level variables using the following cursor:

```
         SELECT NVL(so.stkldgr_vat_incl_retl_ind,'N'),
                so.calendar_454_ind,
                so.currency_code,
                TO_CHAR(sv.last_eom_date,'YYYYMMDD'),
                TO_CHAR(sv.next_eom_date,'YYYYMMDD'),
                TO_CHAR(p.vdate,'YYYYMMDD')
           FROM system_options so,
                system_variables sv,
                period p;
```

The function then calls the function size_arrays() to allocate memory for arrays used in this program.

process()

Call process_actuals_by_order()	(For Bill Back Non Rebate deals)
Call process_actuals_all_orders()	(For Bill Back Purchase Rebate Deals)
Call process_actuals_tran_data()	(For Bill Back Sales and Receipts Deals)

process_actuals_by_order()

Retrieve rows from the C_ORDER_DATA cursor in a while loop.

For each row retrieved, if the deal has a threshold UOM that is different to the Standard UOM, call uom_convert() to convert the units between the two.

If a row exists on DEAL_ACTUALS_ITEM_LOC call add_dail_upd_row(), otherwise call add_dail_ins_row().

When the commit point is reached, call insert_dail_rows() and update_dail_rows().

process_actuals_all_orders()

Retrieve rows from the C_ALL_ORDERS cursor in a while loop.

For each row retrieved, if the deal has a threshold UOM that is different to the Standard UOM, call uom_convert() to convert the units between the two.

If a row exists on DEAL_ACTUALS_ITEM_LOC call add_dail_upd_row(), otherwise call add_dail_ins_row().

When the commit point is reached, call insert_dail_rows() and update_dail_rows().

process_actuals_tran_data()

Retrieve rows from the C_TRAN_DATA cursor in a while loop.

For each Item/Loc/Deal/Deal Detail retrieved, accumulate sales and returns (removing VAT from tran code 4 records if STKLDGR_VAT_INCL_RETL_IND=Y).

If the deal has a threshold UOM that is different to the Standard UOM, call uom_convert() to convert the units between the two.

If a row exists on DEAL_ACTUALS_ITEM_LOC call add_dail_upd_row(), otherwise call add_dail_ins_row().

When the commit point is reached, call insert_dail_rows() and update_dail_rows().

add_dail_ins_row()

This function adds a new element to the pa_ins_item_loc array whilst ensuring that the array size is not exceeded and if necessary resizing the array when required. If the array exceeds the maximum size, resize it before adding the new row by calling the function resize_arrays(). Finally this function adds the new data to the array and increments the array count.

insert_dail_rows()

This function inserts records into the deal_actuals_item_loc database table at two stages within the function process(), when the commit point is reached and also before closing the driving cursor. Upon completion of this function the insert count is reset to zero.

add_dail_upd_row()

This function adds a new element to the pa_upd_item_loc array, whilst ensuring that the array size is not exceeded and if necessary resizing the array when required. If the array exceeds the maximum size, resize it before adding the new row by calling the function resize_arrays(). Upon completion of this function add the new data to the array and update the array count.

update_dail_rows()

This function updates records in the deal_actuals_item_loc database table at two stages within the function process(), when the commit point is reached and also before closing the driving cursor. Upon completion of this function the update count is reset to zero.

uom_convert()

Call database package function UOM_SQL.CONVERT to convert the input Item Qty from one one UOM to the other.

size_arrays()

Allocate memory for elements of the structures used in the program.

resize_arrays()

Use the memory allocation macro to allocate memory for the elements of the structures used in the program based upon the parameter passed to this function.

free_arrays()

Uses the memory deallocation macro to free the memory used by the elements of the structures used in the program.

final()

This function calls the function free_arrays() to free all arrays.

Call standard retek close function retek_close( ) to tidy up restart / recovery.

Input Specifications

Cursor C_ORDER_DATA (used in process_actuals_by_order()):

```
        SELECT dbnrt.order_no,
               dbnrt.deal_id,
               dbnrt.deal_detail_id,
               dbnrt.dai_id,
               dbnrt.item,
               dbnrt.loc_type,
               dbnrt.location,
               dbnrt.order_currency_code,
               dbnrt.loc_currency_code,
               dbnrt.total_units,
               dbnrt.total_revenue,
               dbnrt.reporting_date,
               dh.threshold_limit_uom
          FROM deal_bb_no_rebate_temp dbnrt,
               deal_head dh
         WHERE dh.deal_id = dbnrt.deal_id
           AND MOD(dbnrt.deal_id, TO_NUMBER(:ps_num_threads)) + 1 =
   TO_NUMBER(:ps_thread_val)
           AND (   dbnrt.deal_id   >
   NVL(TO_NUMBER(:ps_restart_deal_id), -999)
                 OR (   dbnrt.deal_id = TO_NUMBER(:ps_restart_deal_id)
                        AND
                        dbnrt.deal_detail_id >
   NVL(TO_NUMBER(:ps_restart_deal_detail_id), -999)
                    )
               )
      ORDER BY dbnrt.deal_id,
               dbnrt.deal_detail_id;
```

Cursor C_ALL_ORDERS (used in process_actuals_all_orders()):

```
        SELECT dbrpt.deal_id,
               dbrpt.deal_detail_id,
               dbrpt.dai_id,
               dbrpt.item,
               dbrpt.loc_type,
               dbrpt.location,
               dbrpt.order_currency_code,
               dbrpt.loc_currency_code,
               dbrpt.total_units,
               dbrpt.total_revenue,
               dbrpt.reporting_date,
               dh.threshold_limit_uom
          FROM deal_bb_rebate_po_temp dbrpt,
               deal_head dh
         WHERE dh.deal_id = dbrpt.deal_id
           AND MOD(dbrpt.deal_id, TO_NUMBER(:ps_num_threads)) + 1 =
   TO_NUMBER(:ps_thread_val)
           AND (   dbrpt.deal_id   >
   NVL(TO_NUMBER(:ps_restart_deal_id), -999)
                 OR (   dbrpt.deal_id = TO_NUMBER(:ps_restart_deal_id)
                        AND
                        dbrpt.deal_detail_id >
   NVL(TO_NUMBER(:ps_restart_deal_detail_id), -999)
                    )
               )
      ORDER BY dbrpt.deal_id,
               dbrpt.deal_detail_id;
```

Cursor C_TRAN_DATA:

```
        SELECT dtdt.deal_id,
               dtdt.deal_detail_id,
               dtdt.item,
               dtdt.loc_type,
               dtdt.location,
               dtdt.dai_id,
               dtdt.tran_code,
               dtdt.reporting_date,
               dtdt.vat_rate,
               dtdt.units,
               dtdt.total_revenue,
               dh.threshold_limit_uom
         FROM deal_tran_data_temp dtdt,
              deal_head dh
        WHERE dh.deal_id = dtdt.deal_id
          AND (   dtdt.deal_id   > NVL(TO_NUMBER(:ps_restart_deal_id),
-999)
```

```
                 OR (   dtdt.deal_id = TO_NUMBER(:ps_restart_deal_id)
                        AND
                        dtdt.deal_detail_id >
   NVL(TO_NUMBER(:ps_restart_deal_detail_id), -999)
                    )
               )
      ORDER BY dtdt.deal_id,
               dtdt.deal_detail_id,
               dtdt.reporting_date,
               dtdt.item,
               dtdt.loc_type,
               dtdt.location,
               dtdt.dai_id,
               dtdt.tran_code;
```

Output Specifications

N/A

Scheduling Considerations

Processing Cycle:	Must be run daily otherwise data will be lost and income can not be calculated retrospectively.
Pre-Processing:	salstage.pc
Post-Processing:	N/A
Threading Scheme:	v_restart_deal

Restart Recovery

The dealact.pc batch program has multi-threading capabilities (deal_id) as well as restart/recovery functionality. The logical unit of work for this program is deal_id/deal_detail_id.

Deal explode [dealex]

Design Overview

This batch program will take the information held in DEAL_ITEMLOC and write them to DEAL_ITEM_LOC_EXPLODE.

This should be run nightly before dealinc.pc where there are records that relate to deals that are not at the item location level, these records need to be exploded down to the item/location detail level before copying them to DEAL_ITEM_LOC_EXPLODE.

For all new approved deals (i.e. do not already exist on DEAL_ITEM_LOC_EXPLODE explode all records down to item/location level (if required) and insert into DEAL_ITEM_LOC_EXPLODE

In addition the deal_explode batch will also need to look at the reclass_cost_chg_queue table to take into account any changes in item hierarchy location relationships since records are written here when a new item is approved / reclassified or associated with a new item supplier country location.

The following rules should also apply to reduce the data held on the DEAL_ITEM_LOC_EXPLODE table:

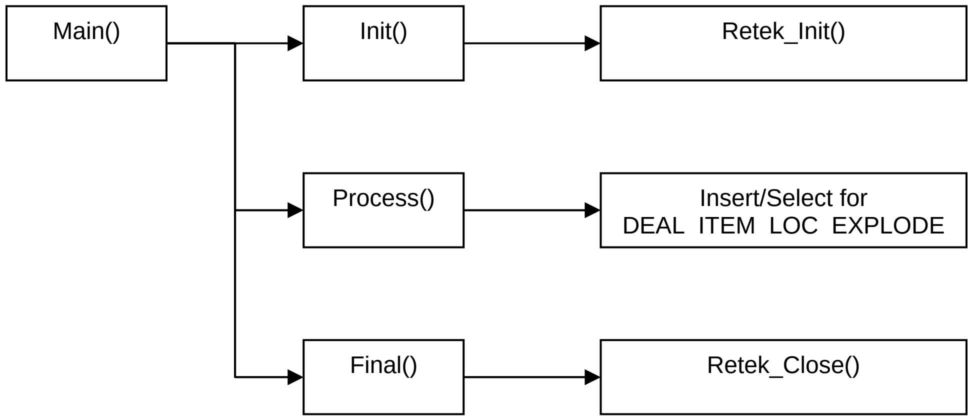
- Non inventory items should not be on this table.
- Transaction-level items only
- In case the deal is receipt or sales based, packs should not be on this table either because we are getting our information from tran_data which does not hold pack information
- If the deal is sales based, only sellable items should be on the table.
- If the deal is receipt based, only orderable items should be on the table.
- If the deal is purchases then it should contain orderable items and packs.
- If the deal is a VFP, off invoice or VFM then we do not need any explosion since posupld and price extraction will create it for us.

Tables Affected:

Table Name	Select	Insert	Update	Delete
DEAL_ITEM_LOC_EXPLODE	Yes	Yes	No	No
V_RESTART_DEAL	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
GROUPS	Yes	No	No	No
DEPS	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
RECLASS_TRIGGER_TEMP	Yes	No	No	No
RECLASS_COST_CHG_QUEUE	Yes	No	No	No

Table Name	Select	Insert	Update	Delete
ITEM_SUPP_COUNTRY	Yes	No	No	No
STORE_HIERARCHY	Yes	No	No	No
WH	Yes	No	No	No
DEAL_ITEMLOC	Yes	No	No	No

Program Flow



Function Level Description

main()

int argc

char *argv[]

Standard RETEK main function.

Log on to DATABASE.

Call Init() to initialize the program.

Call process() to insert records to DEAL_ITEM_LOC_EXPLODE.

Call final() to cleanup.

Log appropriate messages for batch run based on return from above calls.

init()

int argc

char *argv[]

Call standard Retek restart/recovery function retek_init() to populate the ps_num_threads and ps_thread_val restart variables.

Call fetch_vdate() to get the vdate.

process()

No arguments

Checks first if the program has already been run on the current vdate. If so, data has already been written in the table DEAL_ITEM_LOC_EXPLODE and will issue an error message that the program cannot be run more than once on the same date.

```
EXEC SQL DECLARE c_check_date CURSOR FOR
   SELECT /*+ FIRST_ROWS */ 'X'
     FROM v_restart_deal vrd,
          deal_item_loc_explode dile
    WHERE dile.create_date = TO_DATE(:ps_vdate,'YYYYMMDD')
      AND dile.deal_id = vrd.driver_value
      AND vrd.driver_name  = :ps_driver_name
      AND vrd.num_threads  = TO_NUMBER(:ps_num_threads)
      AND vrd.thread_val   = TO_NUMBER(:ps_thread_val)
      AND rownum < 2;
```

Call insert/select statement to select all new approved deals and any new item/locations and reclassified items for insert into the new explode table DEAL_ITEM_LOC_EXPLODE.

The statement will select all new approved deals (as well as other item/location conditions) from the deal tables.

final()

No arguments

Call standard Retek file function retek_close()

fetch_vdate()

No arguments

Populate variable ps_vdate with a call to database function GET_VDATE()

Input Specifications

```
      EXEC SQL INSERT /*+ append */ INTO gtt_dealex_item_master
         SELECT /* parallel(i, 8) */ g.division,
                d.group_no,
                i.dept,
                i.class,
                i.subclass,
                i.item,
                i.status,
                i.pack_ind,
                i.sellable_ind,
                i.orderable_ind,
                i.item_parent,
                i.item_grandparent,
                i.diff_1,
                i.diff_2,
                i.diff_3,
                i.diff_4,
                to_date(NULL) process_date
           FROM groups g,
                deps d,
                item_master i
          WHERE d.dept          = i.dept
            AND g.group_no      = d.group_no
            AND i.inventory_ind = 'Y'
            AND i.status        = 'A'
            AND i.item_level    = i.tran_level;
```

Output Specifications

N/A

Scheduling Considerations

Processing Cycle:	Daily
Scheduling Diagram:	N/A
Pre-Processing:	precostcalc.pc
Post-Processing:	dealinc.pc
Threading Scheme:	DEAL_ID

Restart Recovery

The module will run for a complete set of deal/item/location records multithreaded by DEAL_ID. In the current design restart/recovery has not been included due to the nature and complexity of the select statement required to perform the explosion to item/location level.

Performance Considerations

There is a performance concern regarding the size of the insert/select statement. Due to the structure of the DEAL_ITEMLOC table the statement needs to take into account all levels of organization hierarchy and merchandise hierarchy, and also deal exclusions. In order to enhance performance two inline views (DILT and DIMT) will be used as data lookups for the statements. These tables provide existing item/location and item/merch records as well as new item/location and reclassified items.

The complete statement will be used by an ‘insert’ so as to maximize the performance and the dataset will be multithreaded by DEAL_ID.

Deals Forecast [dealfct]

Design Overview

The purpose of this batch module is to maintain forecast periods, deal component totals and deal totals. After determining which active deals need to have forecast periods updated with actuals, the program will then sum up all the actuals for the deal reporting period and update the deal_actuals_forecast table with the summed values and change the period from a forecast period to a fixed period. The program will also adjust either the deal component totals (deal_detail) or the remaining forecast periods (deal_actuals_forecast) to ensure that the deal totals remain correct. For each deal, the program will also maintain values held at deal_head level (e.g. growth rates, etc.)

The program will be run on the same day as salmonth after the dealinc program has completed.

The program will call the following functions from the dealinclib library to maintain deal forecast periods and deal components:

- Update_actual_fixed_totals – Called when the total_actual_fixed_ind from DEAL_DETAIL is set to ‘Y’. This function recalculates and updates the forecast periods in response to a change made to the actual/forecast value in a reporting period to ensure they still match the deal component total. NOTE: If the current actuals exceed the forecast total then all forecasts are set to zero and the total is updated with the sum of the actuals regardless of the fixed indicator being set.
- Update_budget_fixed_totals – Called when the total_budget_fixed_ind from DEAL_DETAIL is set to ‘Y’. This function recalculates and updates the forecast periods in response to a change made to the budget value in a reporting period to ensure they still match the deal component total. NOTE: If the current budgets exceed the forecast total then all forecasts are set to zero and the total is updated with the sum of the actuals regardless of the fixed indicator being set.
- Update_turnover_trend – This recalculates the actual_forecast_trend_turnover column for forecast periods using the passed growth rate percentage and the forecast turnover.
- Forecast_income_calc –This function will calculate income based upon the budget turnover and actual/forecast/trend turnover values from the DEAL_ACTUALS_FORECAST table. The calculation performed will be determined by the deal income calculation type. The results of the calculations will be written to the DEAL_ACTUALS_FORECAST table. If the deal is in Worksheet status, budget_income is updated. If the deal is in Approved status, actual_forecast_income and actual_forecast_trend_income are updated.
- Update_deal_detail_actual_totals – Called when the total_actual_fixed_ind from DEAL_DETAIL is set to ‘N’. This recalculates the deal totals by summing up all the reporting periods, it then updates the DEAL_DETAIL.total_actual_forecast_turnover row totals with the summed values.
- Update_deal_detail_budget_totals – Called when the total_budget_fixed_ind from DEAL_DETAIL is set to ‘N’. This recalculates the deal totals by summing up all the reporting periods, it then updates the DEAL_DETAIL. total_budget_turnover row totals with the summed values.

- Update_total_baseline **-** This recalculates the baseline growth % in response to a change made to the deal totals and updates the DEAL_DETAIL table. If the deal is in Worksheet status, total_baseline_growth_budget is updated. If the deal is in Approved status, total_baseline_growth_act_for is updated.
- Update_forecast_unit_amt **-** This function will update the total_forecast_revenue or total_forecast_units on the DEAL_DETAIL table, determined by the deal's threshold_limit_type: Quantity or Amount, respectively. The calculation will use the total forecast revenue from the table and the passed amt_per_unit parameter.
- Deal_to_date_calcs – This recalculates the deal-to-date budget growth rate, using the SUMs of the actual turnover and budgeted turnover values for actuals only.

Tables Affected:

TABLE	**INDEX**	**SELECT**	**INSERT**	**UPDATE**	**DELETE**
DEALFCT_TEMP	No	Yes	No	No	No
DEAL_DETAIL	No	No	No	No	No
DEAL_ACTUALS_FORECAST	No	No	No	Yes	No
DEAL_ACTUALS_ITEM_LOC	No	No	No	No	No
STORE	No	No	No	No	No
WH	No	No	No	No	No

Stored Procedures / Shared Modules (Maintainability)

Header file included: DEALINCLIB.h using functions: update_actual_fixed_totals, update_budget_fixed_totals, update_turnover_trend, forecast_income_calc, update_deal_detail_actual_totals, update_deal_detail_budget_totals, update_total_baseline, update_forecast_unit_amt, deal_to_date_calcs

Function Level Description

main()

This function will Validate the program arguments and logon to Oracle, call the init() function to initialize restart / recovery and variables, call the process() function to execute main program logic and then call the final() function to clean up all internal processing

init()

This function calls the standard retek initialization function retek_init() to initialize restart/recovery.

It will then retrieve system level variables:

SYSTEM_OPTIONSNS.CURRENCY_CODE ,

PERIOD.VDATE

Call the function size_arrays()

Process()

This function contains the driving cursor which will retrieve details of forecast periods for active deal components that require processing. The cursor will also return a flag indicating if this is the last reporting period for the component, this is required for Pro-Rate processing as the last period for pro-rated deals requires special processing.

Looping through the fetched data, if the deal period has changed, call add_daf_upd_row() to add totals for previous forecast period to pa_upd_daf array and add a new element to the array.

If last_period_ind is ‘N’, call calc_amount_per_unit() to add the reporting period details to the appropriate forecast period update array, and call add_forecast_period_row() to add a new element to the pa_upd_forecast_periods array. The period totals are then reset..

While processing, if the deal changes call add_deal_upd_row() to add a new element to the pa_upd_deal array.

During processing in the loop, when a commit point has been reached, perform update processing and commit the data to the database: call update_daf_data(), update_forecast_periods(), update_deal_components(), and update_deals().

If deal component has changed, update component data by calling calc_amount_per_unit() to add the reporting period details to the appropriate forecast period update array then call add_component_upd_row() to add a new element to the deal component array.

If the location currency is not the same as the deal currency then call the library function convert() to convert the revenue and income.

Once finished loop processing, all valid data is then inserted/updated in the database.

add_daf_upd_row ()

Adds a new element to the update array whilst ensuring that the array size is not exceeded and if necessary resizing the array when required.

update_daf_data()

Array updates the DEAL_ACTUALS_FORECAST table from the pa_upd_daf_data array. Sets actual_forecast_ind = ‘A’, actual_forecast_turnover, actual_forecast_income, actual_income, actual_forecast_trend_turnover, and actual_forecast_trend_income.

add_deal_upd_row()

Adds a new element to the pa_upd_deal array whilst ensuring that the array size is not exceeded and if necessary resizing the array when required.

add_component_upd_row()

Adds a new element to the pa_upd_deal_detail array whilst ensuring that the array size is not exceeded and if necessary resizing the array when required.

add_forecast_period_row()

Adds a new element to the pa_upd_forecast_periods array whilst ensuring that the array size is not exceeded and if necessary resizing the array when required.

update_forecast_periods ()

This function loops through the pa_upd_forecast_periods array and calls dealinclib library function update_actual_fixed_totals() if total_actual_fixed_ind = ‘Y’.

If rebate_ind = ‘Y’, library function forecast_income_calc() is called.

update_deal_components ()

This function loops through the pa_upd_deal_detail array and calls dealinclib library functions update_deal_detail_actual_totals(), update_total_baseline(), and update_forecast_unit_amt().

update_deals ()

This function loops through the pa_upd_deal array and calls dealinclib library functions update_turnover_trend() and forecast_income_calc().

calc_amount_per_unit ()

This function calculates forecast amounts per unit. The unit can be one of two threshold limit types, Q or A.

‘Q’ means that if total actual forecast turnover is zero, then the amount per unit is zero. If the total actual forecast turnover is NOT zero the amount per unit is equal to the total_forecast_revenue divided by total actual forecast turnover.

‘A’ means that if total actual forecast units is zero, then the amount per unit is zero. If the total actual forecast unit is NOT zero the amount per unit is equal to the total_forecast_units divided by total actual forecast turnover

size_arrays ()

Allocate memory for elements of the structures used in the program.

resize_arrays ()

Use the memory allocation macro to allocate memory for the elements of the structures used in the program.

free_arrays ()

Uses the memory deallocation macro to free the memory used by the elements of the structures used in the program.

handle_shared_lib_error ()

Passing in the two parameters, the calling functions name and the function name being called.

Call the function get_lib_error_message()

Call standard retek close function retek_close().

final()

Free all arrays by calling function free_arrays().

Call standard retek close function retek_close().

Input Specifications

Driving cursor:

```
      SELECT daf_rowid,
             deal_id,
             deal_detail_id,
             dh_currency_code,
             threshold_limit_type,
             rebate_ind,
             total_actual_fixed_ind,
             total_forecast_units,
             total_forecast_revenue,
             total_actual_forecast_turnover,
             reporting_date,
             last_period_ind,
             actual_forecast_turnover,
             vloc_currency_code,
             actual_turnover_units,
             actual_turnover_revenue,
             actual_income
        FROM dealfct_temp
       WHERE restart_thread_return(deal_id, TO_NUMBER(:ps_num_threads))
   =
               TO_NUMBER(:ps_thread_val)
         AND deal_id > NVL(:ps_restart_deal_id, -999)
       ORDER BY deal_id, deal_detail_id, reporting_date;
```

Output Specifications

N/A

Scheduling Considerations

Processing Cycle:	Ad hoc on the same day as salmonth.pc.
Pre-Processing:	dealinc.pc
Post-Processing:	N/A
Threading Scheme:	v_restart_deal

Restart Recovery

The Logical Unit of Work (LUW) for the program is deal_id.

Deal Income Calculation Daily – [dealinc]

Design Overview

For complex deals, this program will retrieve deal attributes and actuals data from the deals tables, it will then calculate the income and will update DEAL_ACTUALS_ITEM_LOC rows with the calculated income value. Additionally the program will insert the income value into the TEMP_TRAN_DATA table using the new tran data codes 6 (Deal Sales) and 7 (Deal Purchases).

Deal calculations are done in deal currency but data held on DEAL_ACTUALS_ITEM_LOC table is in location currency, hence if the currencies differ then the values need to be converted to deal currency before calculation and back to location currency after calculation for subsequent updating of the rows. Currency convert routines in the currconv.pc library will be utilized.

Subsequent programs will run to perform forecast processing for active deals and to roll up TEMP_TRAN_DATA rows inserted by the multiple instances of this module and insert/update DAILY_DATA with the summed values and then insert details from TEMP_TRAN_DATA into TRAN_DATA.

Income is calculated via a call to actual_income_calc() in the dealinclib.pc library, this module will retrieve threshold details for each deal component and determine how to perform the calculation i.e. Linear/Scalar, Actuals Earned/Pro-Rate, etc.

Tables Affected:

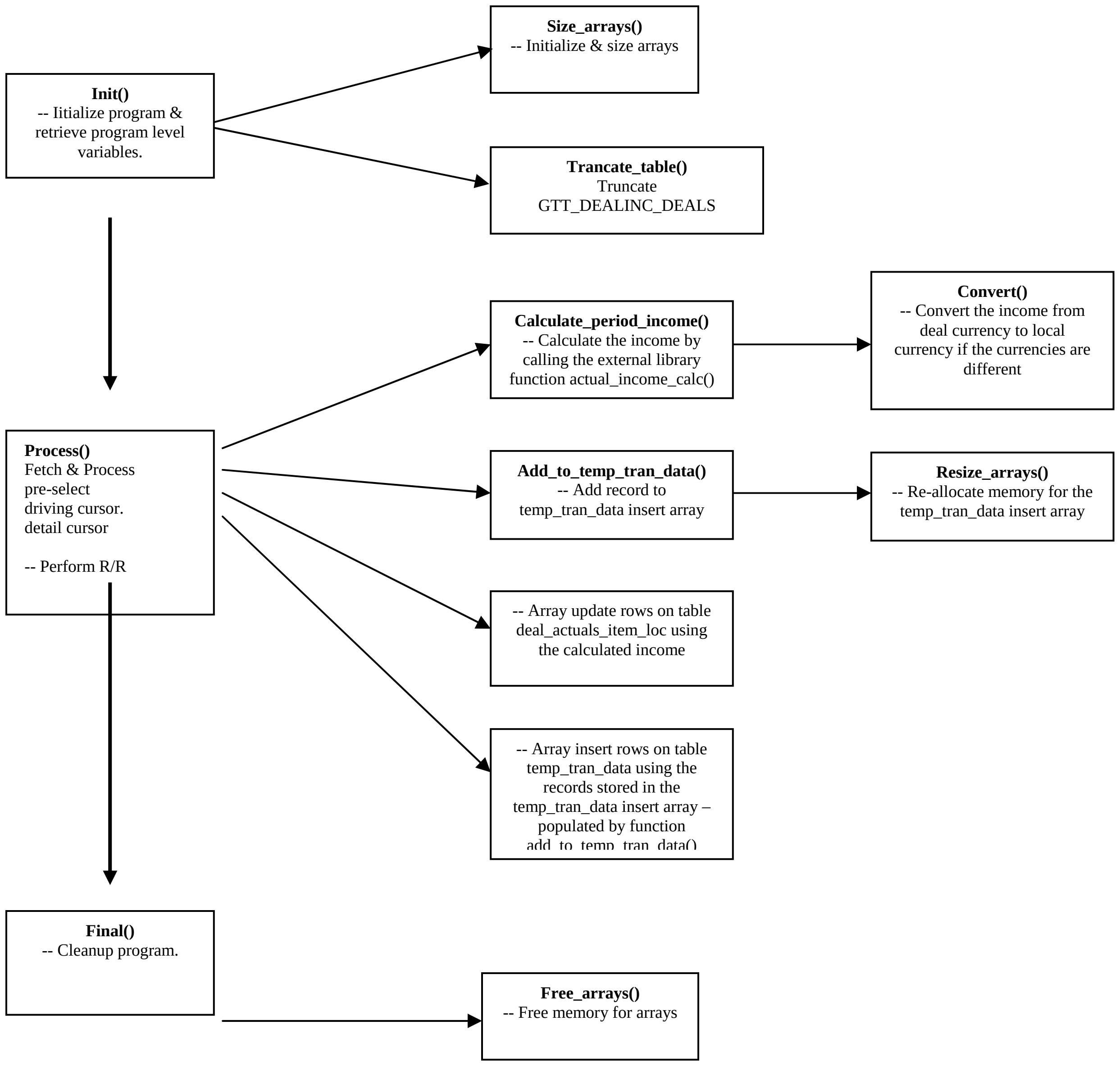
TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
GTT_DEALINC_DEALS	No	Yes	Yes	No	Yes
DEAL_HEAD	Yes	Yes	No	No	No
DEAL_DETAIL	Yes	Yes	No	No	No
DEAL_ACTUALS_ITEM_LOC	Yes	Yes	No	Yes	No
ITEM_MASTER	Yes	Yes	No	No	No
DEAL_ACTUALS_FORECAST	Yes	Yes	No	No	No
TEMP_TRAN_DATA	No	No	Yes	No	No
STORE	Yes	Yes	No	No	No
WH	Yes	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No
PERIOD	No	Yes	No	No	No

Stored Procedures / Shared Modules (Maintainability)

convert (library function)

actual_income_calc (library function)

Program Flow


Init()
-- Iitialize program & retrieve program level variables.
Size_arrays()
-- Initialize & size arrays
Trancate_table()
Truncate GTT_DEALINC_DEALS
Calculate_period_income()
-- Calculate the income by calling the external library function actual_income_calc()
Convert()
-- Convert the income from deal currency to local currency if the currencies are different
Process()
Fetch & Process pre-select driving cursor. detail cursor
-- Perform R/R
Add_to_temp_tran_data()
-- Add record to temp_tran_data insert array
Resize_arrays()
-- Re-allocate memory for the temp_tran_data insert array
-- Array update rows on table deal_actuals_item_loc using the calculated income
-- Array insert rows on table temp_tran_data using the records stored in the temp_tran_data insert array – populated by function add_to_temp_tran_data()
Final()
-- Cleanup program.
Free_arrays()
-- Free memory for arrays

Function Level Description

Init()

EXEC SQL ALTER SESSION SET HASH_AREA_SIZE=104857600;

EXEC SQL ALTER SESSION SET SORT_AREA_SIZE=104857600;

Call standard retek initialization function retek_init() to initialize restart / recovery.

Gets the following system level variables (program variables):

- SYSTEM_OPTIONS.CURRENCY_CODE (ps_primary_currency_code)
- SYSTEM_VARIABLES.LAST_EOM_DATE (ps_last_stkldgr_close_date)
- PERIOD.VDATE (ps_vdate)

Call function size_arrays().

Call function truncate_table(), passing “GTT_DEALINC_DEALS”.

Process()

- Pre-select the deals to be processed into a global temporary table (see “Input Specification” below).
- Commit the inserted records.
- Define the driving cursor.
- Define the detail cursor.
- In a while loop array fetch required information from cursor C_DRIVER.
- For each row retrieved from C_DRIVER, retrieve each row from C_DETAIL in another while loop.
- For each row retrieved call calculate_period_income() to calculate the income using the information retrieved from the driving cursor. The calculated income is written to the corresponding row of the fetch array.
- If the deal currency and the location currency are not the same then the income value will need to be converted back from the deal currency to the location currency as the DEAL_ACTUAL_ITEM_LOC table stores the value in location currency. To do the conversion the library function convert() is used.
- Add the current details to the temp_tran_data insert array using function add_to_temp_tran_data().
- For all rows in the fetch array, an array update is used to update rows on table DEAL_ACTUALS_ITEM_LOC using the information from the driving cursor and the income calculated by the function calculate_period_income(). Care is taken to limit each bulk update to the maximum size defined in MAX_UPDATE_ARRAY_SIZE
- An array insert is used to insert all rows from the temp_tran_data array into table TEMP_TRAN_DATA.. Care is taken to limit each bulk insert to the maximum size defined in MAX_INSERT_ARRAY_SIZE
- For each change of Deal Id/Deal Detail Id, call standard retek function retek_force_commit() to commit the changes to the database.

Calculate_period_income()

- This function will call the library function actual_income_calc() to perform the income calculation for the current period row using the deal details passed into it. If necessary the input values will be converted into the deal currency prior to income being calculated.
- If the deal currency and the location currency are not the same then the actuals value retrieved from DEAL_ACTUALS_ITEM_LOC in the driving cursor need to be converted from the location currency into the deal currency before the income is calculated. To do the conversion the library function convert() is used. This is only required if the limit type is Amount, also the act_for_turnover_total does not need to be converted as it comes from the DEAL_ACTUALS_FORECAST table which is already in the deal currency.
- The amount_per_unit is calculated as follows: When threshold_limit_type is Quantity and threshold_value_type is Percent-Off then the amount_per_unit = actual_turnover_revenue / actual_turnover_units. When threshold_limit_type is Amount and threshold_value_type is Amount-Off then the amount_per_unit = actual_turnover_units / actual_turnover_revenue. In all other cases, the amount_per_unit is defaulted to zero.
- If the deal is prorated and the totals are not fixed, then we need to subtract the current DEAL_ACTUALS_FORECAST.ACTUAL_FORECAST_TURNOVER from actual_forecast_turnover_total as this will become an Actual, when program dealfct.pc runs. The sum of the actual_forecast_turnover is retrieved from table DEAL_ACTUALS_FORECAST for rows where the actual_forecast_ind = ‘F’ (forecast) and the reporting_date <= period.date. This amount is then subtracted from the actual_forecast_turnover_total amount.
- The library function actual_income_calc() is then called using the actual_forecast_turnover_total (if prorated this total will have actual_forecast_turnover already subtracted – see above) and calculated amount_per_unit. All other information is supplied by the driving cursor.

Add_to_temp_tran_data()

- If the temp_tran_data insert array has reached its initial size then need to add another entry to the array using a call to function resize_arrays()
- Copy current record from the driving cursor into the temp_tran_data insert array.

Size_arrays()

- Allocate memory for the driving cursor fetch array and the temp_tran_data insert array.

Resize_arrays()

- Re-allocate memory for the temp_tran_data insert array.

Free_arrays()

- Free memory allocated for the driving cursor fetch array and the temp_tran_data insert array.

Truncate_table()

- Truncate the table name specified by the is_table_name input parameter.

Final()

- Free all arrays by calling function free_arrays().

- Call standard retek close function retek_close().

Input Specifications

Driving cursors:

This pre-select, driving and detail cursors will retrieve active bill back deals rows which require income to be calculated today and the relevant columns from the deal tables to perform this calculation. Active bill back deal periods requiring income calculation are identified as forecast periods where the reporting date <= today.

Pre-Select of Deals to be processed (into GTT_DEALINC_DEALS).

```
      EXEC SQL INSERT INTO gtt_dealinc_deals
         SELECT dh.deal_id,
                dd.deal_detail_id,
                dh.stock_ledger_ind,
                dh.deal_income_calculation,
                dh.threshold_limit_type,
                dd.threshold_value_type,
                dh.rebate_calc_type,
                NVL(dh.currency_code, :ps_primary_currency_code)
currency_code,
                dh.growth_rate_to_date,
                dd.calc_to_zero_ind,
                dd.total_actual_fixed_ind,
                DECODE(dh.rebate_purch_sales_ind, 'P',
:TRAN_CODE_DEAL_PURCHASE,

:TRAN_CODE_DEAL_SALE) rebate_purch_sales_ind,
                daf.reporting_date,
                dh.rebate_ind,
                vdaf.last_reporting_date,
                vdaf.act_for_turnover_total
           FROM deal_head dh,
                deal_detail dd,
                deal_actuals_forecast daf,
                (SELECT /*+ parallel(deal_actuals_forecast, 8) */
                        daf2.deal_id,
                        daf2.deal_detail_id,
                        MAX(daf2.reporting_date) last_reporting_date,
                        SUM(daf2.actual_forecast_turnover)
act_for_turnover_total
```

```
                   FROM deal_actuals_forecast daf2
                  WHERE
   RESTART_THREAD_RETURN(daf2.deal_id,:ps_num_threads) =
   TO_NUMBER(:ps_thread_val)
                  GROUP BY daf2.deal_id, daf2.deal_detail_id) vdaf
          WHERE dh.billing_type          = 'BB'
            AND dh.status                = 'A'
            AND dh.deal_id               = dd.deal_id
            AND dd.deal_id               = daf.deal_id
            AND dd.deal_detail_id        = daf.deal_detail_id
            AND dd.deal_id               = vdaf.deal_id
            AND dd.deal_detail_id        = vdaf.deal_detail_id
            AND daf.reporting_date     <= TO_DATE(:ps_vdate,
   'YYYYMMDD')
            AND daf.reporting_date      >
   TO_DATE(:ps_last_stkldgr_close_date, 'YYYYMMDD')
            AND RESTART_THREAD_RETURN(dh.deal_id,:ps_num_threads) =
   TO_NUMBER(:ps_thread_val)
            AND (dd.deal_id   > NVL(TO_NUMBER(:ps_restart_deal_id), -
   999)
             OR (dd.deal_id = TO_NUMBER(:ps_restart_deal_id) AND
                dd.deal_detail_id >
   NVL(TO_NUMBER(:ps_restart_deal_detail_id), -999)));
```

C_DRIVER.

```
      EXEC SQL DECLARE c_driver CURSOR FOR
         SELECT DISTINCT gdd.deal_id,
                gdd.deal_detail_id
           FROM gtt_dealinc_deals gdd
          ORDER BY gdd.deal_id,
                gdd.deal_detail_id;
```

C_DETAIL.

```
      EXEC SQL DECLARE c_detail CURSOR FOR
         SELECT /*+ ordered */
                gdd.deal_id,
                gdd.deal_detail_id,
                gdd.stock_ledger_ind,
                gdd.deal_income_calculation,
                gdd.threshold_limit_type,
                gdd.threshold_value_type,
                gdd.rebate_calc_type,
                NVL(gdd.currency_code, :ps_primary_currency_code),
                NVL(vloc.currency_code, :ps_primary_currency_code),
                gdd.growth_rate_to_date,
                gdd.calc_to_zero_ind,
                gdd.total_actual_fixed_ind,
                DECODE(gdd.rebate_purch_sales_ind, 'P',
   :TRAN_CODE_DEAL_PURCHASE,

   :TRAN_CODE_DEAL_SALE),
                dail.dai_id,
                dail.item,
                dail.loc_type,
                dail.location,
                TO_CHAR(dail.reporting_date,'YYYYMMDD'),
                NVL(dail.order_no, -1),
                dail.actual_turnover_units,
                dail.actual_turnover_revenue,
                gdd.act_for_turnover_total,
                im.dept,
                im.class,
                im.subclass,
                DECODE(gdd.last_reporting_date,
   dail.reporting_date,'Y','N') last_period,
                gdd.rebate_ind
           FROM gtt_dealinc_deals gdd,
                deal_actuals_item_loc dail,
                item_master im,
                (SELECT st.store loc,
                        st.currency_code,
                        'S' loc_type
                   FROM store st
                  WHERE stockholding_ind = 'Y'
              UNION ALL
                 SELECT wh.wh loc,
                        wh.currency_code,
                        'W' loc_type
                   FROM wh
                  WHERE stockholding_ind = 'Y'
```

```
                    AND finisher_ind = 'N') vloc
          WHERE gdd.deal_id              = TO_NUMBER(:ls_deal_id)
            AND gdd.deal_detail_id       =
   TO_NUMBER(:ls_deal_detail_id)
            AND dail.deal_id             = gdd.deal_id
            AND dail.deal_detail_id      = gdd.deal_detail_id
            AND dail.item                = im.item
            AND dail.location            = vloc.loc
            AND dail.loc_type            = vloc.loc_type
            AND dail.reporting_date     <= TO_DATE(:ps_vdate,
   'YYYYMMDD')
            AND dail.reporting_date      >
   TO_DATE(:ps_last_stkldgr_close_date, 'YYYYMMDD');
```

Output Specifications

N/A

Scheduling Considerations

Processing Cycle:	Ad-Hoc. Must be run before salmth.pc, after dealact.pc and before the new programs which perform forecast processing and DAILY_DATA roll up. The order of the specific modules are: salstage.pc (daily), salapnd.pc (daily), dealex.pc (daily), dealact.pc (daily), salweek.pc/prepost.pc salweek post (weekly), dealinc.pc (monthly), Run dealfct.pc (monhtly), prepost.pc dealday pre (monthly), dealday.pc (monthly), prepost.pc dealday post (monthly), salweek.pc/prepost.pc salweek post (monthly), salmth.pc (monthly)and prepost.pc salmth post (monthly), vendinvc.pc/vendinvf.pc (daily for monthly processing AFTER salmth), (optional prepost vendinv pre if pulling process does not purge tables)
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

The logical unit of work is a transaction comprising deal_id, deal_detail_id. A commit will take place after the number of deals records processed is equal to the max counter from the restart_control table.

Upload customs tariff files [htsupld]

Design Overview

This batch program will be run whenever an updated US customs tariff file is available (probably twice a year) to upload HTS tariff information from the file into RMS HTS tables. The program will handle both the initial HTS information load as well as mid-year HTS updates that are supplied by the US government. The initial upload is handled by inserting information from the file into the tables; updating information already in the tables is handled by adjusting the effective dates of the existing HTS records and inserting a new set of HTS records into the tables.

Updating HTS records should follow the following guidelines:

- No HTS records with the same HTS and import country should have overlapping effect_from and effect_to dates. Import country is passed as an input parameter to the program, so that the program can support different import countries.
- The new HTS effective dates will never chop up the effective dates of an existing HTS, and there will never be any rollback in dates. Therefore, a new HTS can only start in the middle of an existing HTS or cover a completely different time frame after the existing HTS.
- When loading a new HTS that starts in the middle of an existing HTS, the effect_to date of the existing HTS should be adjusted to one day before the new effect_from date.
- No existing HTS information should be purged by the program. It’s the client’s responsibility to handle that.

Tables Affected:

TABLE	SELECT	INSERT	UPDATE	DELETE
HTS	Yes	Yes	Yes	Yes
HTS_TAX	No	Yes	Yes	Yes
HTS_FEE	No	Yes	Yes	Yes
HTS_OGA	No	Yes	Yes	Yes
HTS_TARIFF_TREATMENT	Yes	Yes	Yes	Yes
HTS_TT_EXCLUSIONS	No	Yes	Yes	Yes
TARIFF_TREATMENT	Yes	No	No	No
COUNTRY_TARIFF_TREATMENT	Yes	No	No	No
HTS_CHAPTER	Yes	No	No	No
OGA	Yes	No	No	No
UOM_CLASS	Yes	No	No	No
CODE_DETAIL	Yes	No	No	No
QUOTA_CATEGORY	Yes	No	No	No
COUNTRY	Yes	No	No	No

TABLE	SELECT	INSERT	UPDATE	DELETE
HTS_CVD	No	No	Yes	No
HTS_AD	No	No	Yes	No
HTS_REFERENCE	No	No	Yes	No
ITEM_HTS	Yes	Yes	Yes	No
ITEM_HTS_ASSESS	No	No	Yes	No
ORDSKU_HTS	Yes	Yes	Yes	Yes
MOD_ORDER_ITEM_HTS	No	Yes	No	No
PERIOD	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No
DUAL	Yes	No	No	No
ORDSKU_HTS_ASSESS	No	No	No	Yes
ORDHEAD	Yes	No	No	No
ORDLOC	Yes	No	No	No
ORDSKU	Yes	No	No	No
CE_CHARGES	Yes	No	No	Yes
CE_ORD_ITEM	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
ORDSKU_TEMP	Yes	No	No	No

Stored Procedures / Shared Modules (Maintainability)

ITEM_HTS_SQL.DELETE_ASSESS – given the item, hts, import_country_id,

origin_country_id, effect_to and effect_from, this function deletes the corresponding record from item_hts_assess.

ITEM_HTS_SQL.DEFAULT_CALC_ASSESS – given the item, hts, import_country_id,

origin_country_id, effect_to and effect_from, this function inserts into item_hts_assess, it also will potentially call other package functions and update other tables.

LC_SQL.DELETE_LCORDAPP – given the order_no, this function deletes from lc_ordapply table.

OTB_SQL.ORD_UNAPPROVE – given the order_no, this function updates the otb table.

ITEM_ATTRIB_SQL.GET_STANDARD_UOM – given the item_no, item_type and indicator, this function returns the standard_uom, standard_class, and conv_factor.

UOM_SQL.CONVERT – given the to_uom, from_value, from_uom, item, supplier and

origin_country, this function returns the to_value.

SQL_LIB.BATCH_MSG – returns error message information.

ORDER_HTS_SQL.DELETE_ASSESS -- given the order_no and seq_no, this function deletes from the ordsku_hts_assess table.

ORDER_HTS_SQL.DEFAULT_CALC_ASSESS -- given the order_no, seq_no, pack or item, hts, import_country_id, origin_country_id, effect_to and effect_from, this function inserts into ordsku_hts_assess, it also will potentially call other package functions and update other tables.

CE_CHARGES_SQL.INSERT_COMPS – given the ce_id, vessel_id, voyage_flt_ind, order_no, item, pack_item, hts, import_country_id, effect_from, effect_to, cvb_code this function inserts into the ce_charges table.

Function Level Description

main()

Standard Retek main function. This program takes in four parameters: userid/passwd, input file, reject file, import country id.

init()

- A global variable is used to hold the import country id that is passed in as a program input parameter. Call check_country to make sure that import country exists on the COUNTRY table; return with fatal error if not. It is used as the import country throughout the program.
- Open input file for read and open reject file for write.
- Call retek_init() for restart/recovery initialization.
- If it is a fresh start, call retek_get_record to read the FHEAD line into the fhead structure.
- Fetch vdate from period table.
- Fetch max_item from hts table and max ct from ordsku_hts and max ct from ce_charges.
- Fetch update_item_hts_ind and update_order_hts_ind from the system_options table
- Call check_spi to make sure that ‘C1’ and ‘C2’ exist in the TARIFF_TREATMENT table as SPI’s. ‘C1’ and ‘C2’ are default tariff treatments for every HTS. Return with fatal error if not.

file_process()

- Call function retek_get_record in a while loop to read the THEAD line into the thead structure:
- if the record type returned is ‘FTAIL’, exit the loop; set a save point.
- If the record type returned is ‘THEAD’, read the THEAD line into the thead structure that contains V1, V2, V3, V4 fields. The V4 record is not currently used in RMS/RTM.
- If the record type returned is other than ‘FTAIL’ or ‘THEAD’, give a fatal error (wrong record type).
- Call function process_THEAD to further process data contained in the THEAD. Set process error flag to indicate non-fatal process error.

- Call function retek_get_record in a while loop to read the TDETL line into the tdetl structure:
  - if the record type returned is ‘TTAIL’, exit to the outer loop to continue reading THEAD records if any exists;
  - if the record type returned is other than ‘TDETL’ or ‘TTAIL’, give a fatal error (wrong record type).
- Call function process_TDETL to further process data contained in the TDETL.
- Set process error flag to indicate non-fatal process error.
- If update_item_hts_ind = “Y”,
  - If tran_code is “A” or “R”, call item_hts_update function. “A” stands for Update only and “R” stands for Replace. In both of these cases (as opposed to the other possibility of “D” for Delete) item tables will need to be updated.
- If update_order_hts_ind = “Y”, call ordsku_hts_search function.
- If process error flag is set. Rollback database process to the save point. Write rejected records to the reject file.
- Call restart_force_commit to perform intermittent commit for restart/recovery.

process_THEAD()

- Fill the hts_keys structure with data from THEAD.
- After filling in the hts_keys, verify that effect_from < effect_to date. If not, reject the record right away. Call valid_all_numeric function to check effect_from, and effect_to field. If invalid reject the record. This function processes the information in V1, V2 and V3 records based on the transaction code ("A","R", "D") in the V1 record. It compares the new effective dates against those of any existing HTS records with the same HTS code and import country.
- If the transaction code type is ‘A’, insert a record into the HTS table; if the transaction code type is ‘R’, update the HTS record that has the same HTS code, import country id, effect_from and effect_to dates
- For transaction code "A":
  - If new HTS covers a time period different than and after any existing HTS, or no HTS exists for the given HTS/import country, is a valid record for inserting.
  - If new the HTS record is overlapping with existing record and its effect_from date > existing record and effect_to >= existing effect_to date, it is a valid record. Process is as follows:
    - Insert an HTS record with the same data as the existing overlapping HTS, except that the effect_to date should be 1 day before the effect_from date of the new HTS record.
    - Update the effect_to date of all corresponding child records to 1 day before the effect_from date of the new HTS record. Insert new hts to the related tables.

Detailed technical description:

- Call function validate_hts_update to verify that the record is valid for insert/update to the database or reject to the reject file. For the valid record call hts_child_update function to prepare child table processing.
- Call hts_table_insert function to insert record to the hts table. if any invalid information exists, write to error file.
- Call hts_oga_insert function to insert record/s to the hts_oga table. if any invalid information exists, write to error file.
- Call hts_spi_insert function to insert record/s to the hts_tariff_treatment table. if any invalid information exists, write to error file.
- Call hts_gsp_insert function to insert record/s to the hts_tt_exclusions table. if any invalid information exists, write to error message log file.
- Set process error flag if non fatal error occurs. Return error flag.
- For transaction code "R":
  - Search for the HTS with the same HTS, import country id, effect_from and effect_to dates. If no record found, reject the record.
  - If a record is found, delete the following child table records with the same HTS, import country id, effect_from and effect_to dates.
  - Insert to update the HTS table and re-insert child table information from the input file.
- Detailed Techincal Description:
  - Call function search_hts_update to find record that can be updated in the database tables.
  - If one exists, prepare child tables for processing.
  - Call hts_table_insert function to insert record to the hts table. if any invalid information exists, write to error file.
  - Call hts_oga_insert function to insert record/s to the hts_oga table. if any invalid information exists, write to error file.
  - Call hts_spi_insert function to insert record/s to the hts_tariff_treatment table. if any invalid information exists, write to error file.
  - Call hts_gsp_insert function to insert record/s to the hts_tt_exclusions table. if any invalid information exist, write to error message log file.
- Set process error flag if non fatal error occurs. Return error flag.
- For transaction code "D":
  - Seach for the HTS with same HTS, import country id , effect_from and effect_to dates.
  - If a record is found update HTS and all its child records to yesterday.

  Note: since the dates are still presented in 2-digit year in the 99 tape, we assume that the year coming in as 00-49 means 2000-2049, and 50-99 means 1950-1999. The customs uses ‘999999’ to mean Dec 31 st , 2039.

Detailed Technical Description:

- Call function search_hts_reset to find updateable record in the hts table. If one exists, insert new hts record.
- Call function hts_child_update to update all the child records, then delete the existing hts record.

validate_hts_update()

- Call out c_hts_date_invalid cursor to select HTS records which starts before or on the same day as any existing HTS, or starts after and ends before any existing HTS:
  - effect_from >= new effect_from OR
  - effect_from < new effect_from and effect_to > new effect_to

If record exists:

- Call out c_hts_date_invalid2 cursor to select HTS records which starts before any existing HTS and ends on Dec 31 st , 2039.
- If record is not found, Write the record to the reject file, write an error message to the message log file, and return to the calling function with a non-fatal error.

  Else, set indictor =true (so that the existing record will be truncated to end 1 day before new HTS starts).
- New HTS starts after and overlaps with an existing HTS:
  - effect_from < new effect_from and effect_to >= new effect_from or new HTS starts after old end date and therefore does not overlap at all. The ranges are completely separate.

    This is a valid record, and a most likely scenario. Fetch the effect_from and effect_to of the existing HTS. Insert a new record with effect_from date same as existing overlapping hts record and effect_to date is 1 day before the new effect_from date to hts table.
  - Call function hts_child_update function to update effect_to date of all child records to 1 day before the new effect_from date.
  - Delete the old record from hts table.

search_hts_update()

- Search for the HTS with the same HTS, import country id, effect_from and effect_to dates. If no record found, reject the record.
- If a record is found, delete the following child table records with the same HTS, import country id, effect_from and effect_to dates:
  - HTS_TT_EXCLUSIONS
  - HTS_TARIFF_TREATMENT
  - HTS_OGA
  - HTS_TAX
  - HTS_FEE

> **Note:** HTS table record cannot be deleted due to the other child tables on HTS: ITEM_HTS, ITEM_HTS_ASSESS, ORDSKU_HTS, HTS_CVD, HTS_AD, HTS_REFERENCE, HTS_CHAPTER. The information on these tables won't be loaded in the HTS upload process.

search_hts_reset()

- Search for the HTS with the same HTS, import country id, effect_from and effect_to dates. If no record found, reject the record.
- Insert into HTS, all the same information, but inserting yesterday as the new to_date.
- If a record is found, call hts_child_update function to update the records in the child tables with effect_to date to yesterday:

hts_child_update()

This function updates the effect_to date of the existing overlapping HTS record on child tables. Since the child tables have referential constraints on the effective dates of the parent table HTS.

- Update the effect_to date of all corresponding child records to 1 day before the effect_from date of the new HTS record.

  The following child tables should be updated:

  - HTS_TARIFF_TREATMENT
  - HTS_TT_EXCLUSIONS
  - HTS_AD
  - HTS_CVD
  - HTS_OGA
  - HTS_REFERENCE
  - HTS_TAX
  - HTS_FEE
  - ITEM_HTS
  - ITEM_HTS_ASSESS
  - ORDSKU_HTS
  - CE_CHARGES

  > **Note:** Since table HTS_TT_EXCLUSIONS has a foreign key on the effect_to date of table HTS_TARIFF_TREATMENT, we cannot update the effect_to date of HTS_TARIFF_TREATMENT directly. Likewise, insert an HTS_TARIFF_TREATMENT record with the new effect_to date first; then update the effect_to date of the HTS_TT_EXCLUSIONS table; at the end delete the HTS_TARIFF_TREATMENT record with the original effect_to date.

- Call delete_ord_temp_tables and pass in the value "-1" because there is no known order_no at this point.

item_hts_update()

- Call size_item_array function to allocate space for the items
- Fetch item, origin_country_id and status from item_hts into struct
- If no data found, call free_itemlist and go to the next record. If data is found, Loop
  - If tran_code = “A” the item will need to be inserted with the same data as the fetched record but with new effect_to and effect_from dates.
    - Insert dates into item_hts.
    - Delete old record from item_hts
    - Call the package SQL Delete assess to delete the old records from item_hts_assess.
  - If tran_code = “R”
    - Call SQL Delete_assess to delete the old records from item_hts_assess
    - Call SQL Default_calc_assess to update the item_hts_assess table (ie insert record with new dates and recalculate)
    - Call ECL_CALC_SQL.CALC_COMP to recalculate expenses based on new assesses.
    - Insert into mod_order_item_hts a new record with same data but new dates.
    - Call free_itemlist

ordsku_hts_search()

Call size_ord_array function to allocate space for the order information

- Fetch values from ordhead, ordsku_hts, ordsku and ordloc into struct (all necessary values to be able to do a complete insert into the mod_order_item, ordsku_hts, and ordsku_hts_assess tables.
- If no data found, call free_ordlist and go to next record. If data is found, Loop
  - If order status = “A”, (the order needs to be updated) set status from approved back to worksheet by calling SQL functions (LC_SQL.DELETE_LCORDAPP and
  - OTB_SQL.ORD_UNAPPROVE).
  - Insert into mod_order_item_hts table (just the order_no and indicator set to ‘Y’)
  - Call ordsku_hts_update
  - Call free_ordlist

ordsku_hts_update()

- Call size_ce_array to allocate space for the custom entry information
  - Fetch custom entry values from ce_ord_item, ce_head, item_supp_country into struct
  - If no data found, call ordhts_update. If data is found:
    - If CE status = “W”, (worksheet status)
    - Call ordhts_update
    - Loop for each custom entry record

    - Call ce_update
        - If status != “W” then the quantity cleared will need to be compared to the total quantity. In order to do that they will need to be converted to the standard uom format
        - Loop
- Call uom_convert to get the total quantity.
- If total_qty < qty_ordered
- Call ordhts_update
- Call free_ceordlist

ordhts_update()

- If tran_code = “A” or “D”
    - Delete old record (record with old dates) from ordsku_hts_assess
    - Delete old record (record with old dates) from ordsku_hts
    - Insert record with new dates into ordsku_hts
- Else if tran_code = “A”
    - Insert record with new dates into ordsku_hts
- Else if tran_code = “D”
    - Call SQL Delete_assess by calling order_del_assess function
    - Call SQL calc_comp
    - If the item is a pack item check to see if a record already exists on mod_order_item_hts – if it does not, insert one with the pack_item
    - If it is not a pack item, insert with item_no into mod_order_item_hts.
    - Return 0
- Else if tran_code = “R”
    - Call delete_ord_temp_tables and pass in the order_no.
- Call SQL Delete_assess by calling order_del_assess function
- Call ORDER_HTS_SQL.DEFAULT_CALC_ASSESS with either the pack_no or item_no depending on if it is a pack or not.
- Call ELC_CALC.CALC_COMP
- If it is a pack item insert into mod_order_hts with the pack_no

If it is not a pack item, insert into mod_order_item_hts with the item_no ce_update()

- Delete from ce_charges.
- If it is a “D”, call CE_CHARGES_SQL.INSERT_COMPS

hts_table_insert()

Before inserting into or updating the HTS table,

- Call function check_chapter to make sure that the chapter already exists on the
- HTS_CHAPTER table. If not, reject the record;
- Call check valid_all_numeric function to check unit for all numeric value.
- Call function check_uom to make sure that the UOMs (UOM1, UOM2, UOM3) already exist on the UOM_CLASS table. Reject the record if UOM does not exist.
- Call function check_duty to make sure that the duty code already exists on the
- CODE_DETAIL table. If not, reject the record.
- Call valid_all_numeric function to verify that the quota is all numeric. Then calling function check_quota to make sure that the quota category already exists on the
- QUOTA_CATEGORY table. If not, reject the record.

Update the existing hts record with the updated hts_desc, chapter, units, units_1, units_2, units_3, duty_comp_code, more_hts_ind, quota_cat, quota_ind, ad_ind, cvd_ind.

Insert the following into the HTS table:

- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- hts_desc: commodity description (V1l)
- chapter: 1 st 4 (leftmost) digits of tariff number
- units: number of reporting units (V1g)
- units_1: first unit of measure (V1h) (If the number of reporting units is zero, this should
- be defaulted to ‘X’)
- units_2: second unit of measure (V1I) –NULL if not given
- units_3: third unit of measure (V1j)—NULL if not given
- duty_comp_code: duty code (V1k)
- more_hts:Y if additional tariff indicator (V2j is ‘R’, N otherwise
- quota_cat: category number (V3h) but only if quota indicator (V3g) is 1
- quota_ind ‘Y’ if there is a quota,’N’ otherwise
- ad_ind ‘Y’ if the anti-dumping flag (V3f) is 1, N otherwise

- cvd_ind 'Y'if the countervailing duty flag (V2k) is 1, N otherwise

hts_oga_insert()

For each OGA code, call function check_oga to verify that the OGA code exists on the OGA table. If not, reject the record; otherwise, call hts_oga_insert to insert into HTS_OGA.

- Insert the following into the HTS_OGA table:
- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- code: OGA code from OGA codes field (V3f)

reference_id: NULL

- comments: NULL

hts_spi_insert()

For each SPI, call function check_spi to check if the SPI exists on the tariff_treatment table; if not, reject the record. Call function hts_tariff_treatment_insert to insert into HTS_TARIFF_TREATMENT. In addition to the SPI records in V3, 'C1' and 'C2' are default tariff_treatments for every HTS. So, two extra records should be inserted into HTS_TARIFF_TREATMENT with SPI codes 'C1' and 'C2'. 'C1' takes the special_duty_rate from V1 and Column 1 rates from V2; 'C2' takes Column 2 rates from V2. Before inserting, call function check_spi to make sure that the SPI code (tariff treatment) exists on the TARIFF_TREATMENT table; reject the record if it does not.

Call valid_all_numeric function to check specific_rate, ad_rate, other_rate for all numeric value. If not, reject the record.

Reject HTS lines that have rate greater than 9999999999. A brief explanation of why this is done is located at the end of the function level description section.

Insert the following into the HTS_TARIFF_TREATMENT table:

- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- tariff_treatment: SPI code from V3i
- specific_rate: 0,col1 or col2 specific rate, as appropriate (0 for SPI's,col 1 for col1, col 2 for col2)
- av_rate: 0,col1, or col2 ad valorem rate, as appropriate (0 for SPI's)
- other_rate: 0,col1, or col2 other rate, as appropriate (0 for SPI's)

hts_gsp_insert()

For each GSP excluded country, call function check_country_tariff_treatment to check that the country and tariff treatment combination exists on the COUNTRY_TARIFF_TREATMENT table; if not, reject the record.

Insert the following into the HTS_TT_EXCLUSIONS table

- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- tariff_treatment: first SPI code from V3i
- origin_country_id: excluded country code from V3d (GSP excluded countries)

check_spi()

Check to see if SPI exists on TARIFF_TREATMENT table; reject the record if it doesn’t.

check_country()

Check to see if country exists on COUNTRY table; reject the record if it doesn’t.

check_chapter()

Check to see if chapter exists on the HTS_CHAPTER table and reject the record if it doesn’t.

check_uom()

Check to see if uom exists on UOM_CLASS table; reject the record if it doesn’t.

check_duty()

Check to see if duty code exists on CODE_DETAIL table (check for the code where

code_type=’DCMP’); reject the record if it doesn’t.

check_quota()

Check to see if the quota_category exists on the QUOTA_CATEGORY table; reject the record if it doesn’t.

check_oga()

Check to see if the oga code exists on the OGA table; reject the record if it doesn’t.

check_comb_country_tt()

Check to see if the country and tariff_treatment combination exists on the

COUNTRY_TARIFF_TREATMENT table; reject the record if it doesn’t.

process_TDETL()

- Format the tax line information from tdetl structure.
- Call function process_taxfees, if no non-fatal error in the process_THEAD function.

process_taxfees()

If tax specific rate or tax ad rate is not null, call hts_taxfee_insert to insert the tax rates into HTS_TAX or HTS_FEE tables. If special rates exist on the tax line, call function hts_tariff_treatment_insert to insert into the HTS_TARIFF_TREATMENT table using the ISO country code as the tariff treatment (SPI). If the SPI given on the tax line already exists for the HTS, the record should be updated, as the tax line special rate takes precedence over the V3 line SPI's rate

Call valid_all_numeric function to check tax_specific_rate, tax_av_rate, fee_specific_rate, fee_av_rate for all numeric value, if not reject the record.

Reject HTS lines that have rate greater than 9999999999. A brief explanation of why this is done is located at the end of the function level description section.

hts_taxfee_insert()

If the tax class code is 016,017,018,or 022 it is a tax; insert into HTS_TAX

If the tax class code is 038,053,054,055,056,057,079,090,103 it is a fee; insert into HTS_FEE

Insert the following into the HTS_TAX or HTS_FEE table:

- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- tax_type/fee_type: tax class code (V5h)
- tax_comp_code/fee_comp_code: tax comp code (V5i)
- tax_specific_rate/fee_specific_rate: tax specific rate (V5k)

tax_av_rate/fee_av_rate: tax ad valorem rate (V5l)

hts_tariff_treatment_insert()

Before calling this function, call function check_spi to make sure that the SPI code (tariff treatment) exists on the TARIFF_TREATMENT table; reject the record if it does not.

Insert the following into the HTS_TARIFF_TREATMENT table:

- hts: tariff number (V1c)
- import_country_id: import country from the program input parameter
- effect_from: begin effective date (V1e)
- effect_to: end effective date (V1f)
- tariff_treatment: SPI code from V3i
- specific_rate: 0,col1 or col2 specific rate, as appropriate (0 for SPI's,col 1 for col1, col 2 for col2)
- av_rate: 0,col1 or col2 ad valorem rate, as appropriate (0 for SPI's)
- other_rate: 0,col1 or col2 other rate, as appropriate (0 for SPI's)

size_item_array()

Allocates space for the item array struct size_ord_array()

Allocates space for the order array struct

size_ce_array()

Allocates space for the custom entry array struct

free_orditemlist()

Frees the space in the array

free_itemlist()

Frees the space in the array

free_ceordlist()

Frees the space in the array

uom_convert()

- Calls ITEM_ATTRIB_SQL.GET_STANDARD_UOM
- Calls UOM_SQL.CONVERT

order_del_assess()

- Calls ORDER_HTS_SQL.DELETE_ASSESS

delete_ord_temp_tables()

If an order no is not passed in, look at the hts table and see if there is an order that exists for that hts. If so, loop and for each record see if there is a record to delete on the temp tables by calling ORDER_SETUP_SQL.DELTE_TEMP_TABLES.

If the order number was passed in, call ORDER_SETU_SQL.DELETE_TEMP_TABLES right away.

final ()

Restart/recovery close and close input and reject file.

Why HTS lines that have a rate greater than 9999999999 need to be rejected:

For fields specific_rate, av_rate, other_rate, RMS has the data type Number(12,8) and numbers coming in from the customs tape also have 8 implied digits. However, when storing the number into the Retek database, we need to divide the number coming in from the customs tape by 1000000 (left shift 6 digits) instead of 100000000 (left shift 8 digits). This is because Retek stores the percent part of the rate only. In other words, rate 11.5% (0.115) is stored as 11.5 in Retek database, whereas it will come in from the customs tape as 11500000 (=0.115). Therefore, the highest rate that can be represented in Retek is 9999.99999999% (= 99.9999999999, or < 100 times). So we need to reject HTS lines that have rate greater than 9999999999.

Note: This is true for hts spi and hts tax/fee specific_rate and av_rate, except that when 999999999999

Input Specifications

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
	Retek file ID	Char(5)	HTSUP	Describes file type
THEAD	Record Descriptor	Char(5)	THEAD	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction id	Number(14)		Unique transaction id
	HTS Line	Char(320)		V1 through V4 records from the customs HTS file concatenated together
TDETL	Record Descriptor	Char(5)	TDETL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction id	Number(10)		Unique transaction id
	Tax/fee line	Char(80)		V5 through V9 records from the customs HTS file, each on a separate TDETL line
TTAIL	Record Descriptor	Char(5)	TTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Detail lines	Number(6)		Number of lines between THEAD and TTAIL
FTAIL	Record Descriptor	Char(5)	FTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction Lines	Number(10)		Number of lines between FHEAD and FTAIL

Output Specifications

N/A

Scheduling Considerations

Processing Cycle:	Ad hoc
Scheduling Diagram:	Run anytime as needed.
Pre-Processing:	after hts upload conversion (hts2rms – PERL script).
Post-Processing:	None
Threading Scheme:	None

Restart Recovery

This program supports Retek standard intermittent commit and file upload restart/recovery. Recommended commit counter is 2000 (commit after every 2000 tariff records are read). Input file names must end in a “.1” for the restart mechanism to properly parse the file name. Since there is only 1 input file to be uploaded, only 1 thread is used. A reject file is used to hold records that have failed processing. The user can fix the rejected records and process the reject file again.

Recommended Order Quantity [ociroq]

Design Overview

The purpose of this batch program is to call the PL/SQL packages used to calculate the Net Inventory position of the items on replenishment. The results are stored in the database to be used by REQEXT (Item Requisition Extraction).

Tables Affected:

TABLE	SELECT	INSERT	UPDATE	DELETE
DOMAIN_CLASS	Y	N	N	N
DOMAIN_DEPT	Y	N	N	N
DOMAIN_SUBCLASS	Y	N	N	N
ITEM_SUPP_COUNTRY	Y	N	N	N
PERIOD	Y	N	N	N
REPL_DAY	Y	N	N	N
REPL_ITEM_LOC	Y	N	N	N
RPL_NET_INVENTORY_TMP	N	Y	N	N
STORE	Y	N	N	N
SYSTEM_OPTIONS	Y	N	N	N
WH	Y	N	N	N
WIN_WH	Y	N	N	N

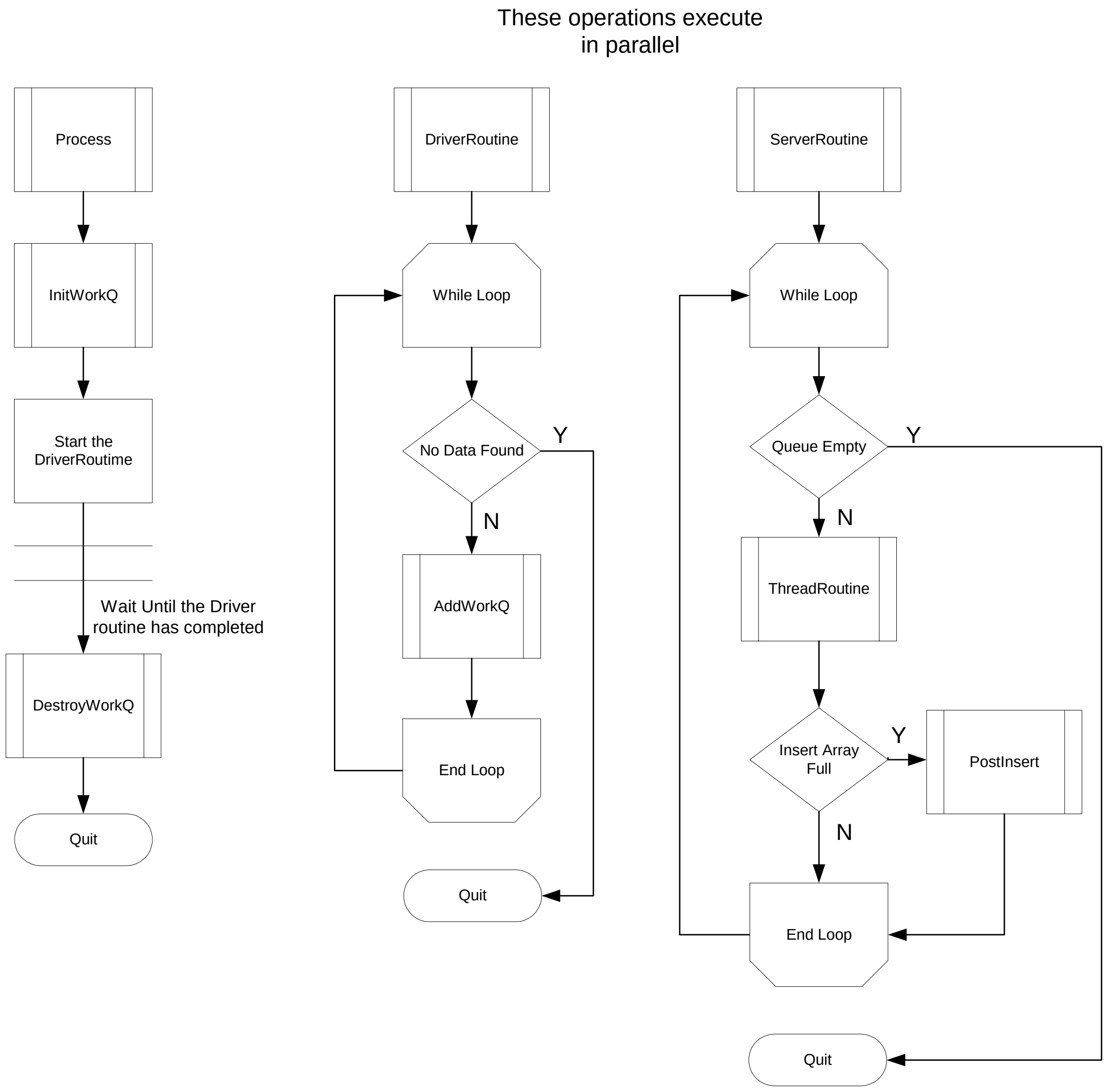
Scheduling Constraints

Processing Cycle:	PHASE 3
Scheduling Diagram:	Prepost (ociroq pre), rplatupd, rpladjf and rpladjs need to run before reqext so that all replenishment calculation attributes are up to date. Posupld needs to run before reqext so that all stock information is up to date.
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	POSIX threads - The restart_control.num_threads will control the number of POSIX threads that are run within ociroq. The batch program ociroq.c itself will only need to be run with one thread.

Restart Recovery

The program processes all items on repl_day for the current day. If the program fails, the rpl_net_inventory_tmp table should be truncated prior to restarting (prepost ociroq pre)

Program Flow



Shared Modules

GET_REPL_ORDER_QTY_SQL.REPL_METHOD: Stored PL/SQL procedure for calculating the ROQ of an item at a location.

REPLENISHMENT_SQL.GET_STORE_REVIEW_TIME: Stored PL/SQL procedure for calculating the time between scheduled shipments to a store from a warehouse. This time is used by GET_REPL_ORDER_QTY_SQL in its calculations.

OciInitLogon(): C library function that validates the program usage and performs initial environment set-up; including opening the daily log file for writing. It also calls OciConnect().

OciConnect(): C library function that connects to the database and performs some initial environment set-up. This function calls numerous OCI library routines that create the appropriate OCI handles.

OciDisconnet(): C library function that disconnects from the database and free the OCI handles created by the OCIConnect() call.

ReportError(): C library function that calls the OCIErrorGet() function and returns the appropriate error message.

WriteError(): C library function writes the appropriate message to the error file; indicating the type of error encountered and the Oracle Error number and message.

LogMessage(): C library function writes the appropriate message to the log file; indicating start time, end time and time of failure if the program terminated with errors.

RaiseError(): C library function responsible for passing the error code back to the parent process to ensure correct error handling.

Data Structures

repl_info_struct: Holds information fetched from the driving cursor.

GetOltsStruct: Holds the information passed into and returned from the REPL_OLT_SQL.GET_OLTS_AND_REVIEW_TIME procedure.

GetReplStruct: Holds the information passed into and returned from the GET_REPL_ORDER_QTY_SQL.REPL_METHOD procedure.

InsertStruct: Used to buffer the inserts into the rpl_net_inventory_tmp table.

DomainStruct: Used to cache forecasting domain information.

Driver_Info: Used by the Driver thread as a container to pass in all the appropriate parameters to the thread routine.

Thread_Info: Used by the Work Queue threads as a container to pass in all the appropriate parameters to the thread routine.

WorkQueue_List: This linked list is used to hold the actual data fetched by the Driver thread to be then consumed by the Work Queue threads.

WorkQueue_Info: Holds all the Work Queue thread control information.

domain_struct: Used to cache forecasting domain information.

Function Level Description

General Controlling Functions

main()

The standard Retek main function, this calls init(), process() and final(), and posts messages to the daily log files.

init()

Fetches system-level global variables and calls other functions to fetch additional global level data; GetNumThreads(), GetStoreCount() and LoadDomainInfo()

Process()

Controls the bulk of the processing. It initializes the Work Queue threads, creates the Driver thread and waits until the Driver thread has completed prior to calling the DestroyWorkQ() and ThreadCleanUp() functions.

final()

The standard Retek final function, this closes down the process and posts messages to the daily logs.

Thread Controlling Functions

InitWorkQ()

Initializes the specific POSIX Pthread library variables used by the Work Queue threads. It then initializes the WorkQueue_Info structure variables and creates the specified number of threads; Each thread calls the ServerRoutine(). The function performs a loop, allocating memory for each threads data structures and connects each thread to the database by calling the OciConnect() library routine. Finally it calls the DefineWorkerStmts() function.

ServerRoutine()

Controls the consumption of the WorkQueue_List. Each Work Queue thread monitors and consumes data from the list until they are instructed to quit or the queue is empty. Initially while the queue is empty the threads poll the queue every 2 seconds checking the status. All thread synchronization is handled by the use of a mutually exclusive lock (mutex). Each node taken from the list is passed to the ThreadRoutine() function.

ThreadRoutine()

Executed by the Work Queue threads; it calls the PL/SQL packages and buffers the result in the InsertStruct. When an individual thread reaches the MAX_INSERT_SIZE the buffer is inserted into the rpl_net_inventory_tmp table.

GetOlts()

Called by the ThreadRoutine(), this function calls the REPL_OLT_SQL.GET_OLTS_AND_REVIEW_TIME PL/SQL package.

GetRepl()

Called by the ThreadRoutine(), this function calls the GET_REPL_ORDER_QTY_SQL.REPL_METHOD PL/SQL package.

DriverRoutine()

Executed by the Driver thread; it's responsible for defining and fetching the driving cursor and adding the batch to the queue. The execution of this function by a thread allows it to run in parallel with the Work Queue threads. The Work Queue threads will start after the first batch has been placed on the WorkQueue_List.

AddWorkQ()

Loads the array fetched by the DriverRoutine() onto the WorkQueue_List. It allocates memory for each node and will continue to load the queue while the number of records on the queue has not exceeded the MAX_QUEUE_SIZE. All thread synchronization is handled by the use of a mutually exclusive lock (mutex). The function will wait until the queue less than half full prior to recommencing.

DestroyWorkQ()

Waits until all the Work Queue threads have consumed all the data from the list; it then performs some cleanup duties. All thread synchronization is handled by the use of a mutually exclusive lock (mutex).

ThreadCleanUp()

Frees the memory allocated to each threads data structures (including statement handles) and disconnects from the database.

Database DML Handling

PostInsert()

When the Insert buffer reaches the MAX_INSERT_SIZE the array is posted to the database and the work committed.

OCI Statement Functions

DefineDriver()

Performs OCI specific statement set-up; including statement handle preparation, statement handle attribute set-up (pre-fetch size), statement column definition and the array of structure definition (skip size etc.) for the Driving Cursor.

DefineGetRepl() **

Performs OCI specific statement set-up; including statement handle allocation, statement handle preparation and statement column binding for the PL/SQL package call GET_REPL_ORDER_QTY_SQL.REPL_METHOD.

DefineGetOlts() **

Performs OCI specific statement set-up; including statement handle allocation, statement handle preparation and statement column binding for the PL/SQL package call REPL_OLT_SQL.GET_OLTS_AND_REVIEW_TIME.

DefineInsert() **

Performs OCI specific statement set-up; including statement handle preparation, statement handle attribute set-up (pre-fetch size), statement column definition and the array of structure definition (skip size etc.) for the rpl_net_inventory_tmp insert Statement.

Note: These functions are called for each Work Queue thread. Each thread will have its own database connection and statement handles.

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.

Stored Procedures / Shared Modules (Maintainability)

validate_all_numeric: intrface library function.

validate_all_numeric_signed: intrface library function.

valid_date: intrface library function.

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

CAL_TO_CAL_LDOM database procedure called from get_eow_eom_date() function

CAL_TO_454_LDOM database procedure called from get_eow_eom_date() function

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

CURRENCY_SQL.CONVERT: returns the converted monetary amount from Currency to currency.

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

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

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

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

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

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

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

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

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
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

main()

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

init()

initialize restart recovery

open input file (posupld)

- file should be specified as input parameter to program

fetch system variables, including the SYSTEM_OPTIONS.CLASS_LEVEL_VAT_IND.

fetch pi_multi_prom_ind from RPM interface

retrieve all valid promotion types and uom class types

fetch uom class types for look up during THEAD processing

declare memory required for all arrays setup for array processing

declare final output filename (used in restart_write_file logic)

open reject file (as a temporary file for restart)

- file should be specified as input parameter to program

open lock reject file (as a temporary file for restart)

- file should be specified as input parameter to program call restart_file_init logic

assign application image array variables- line counter (g_l_rec_cnt), reject counter (g_l_rej_cnt), lock reject file counters (pl_lock_cnt, pl_lock_dtl_cnt), store, transaction_date

if fresh start (l_file_start = 0)

read file header record (get_record)

write FHEAD to lock reject file

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

validate file type = ‘POSU’

else fseek to l_file_start location

validate location and date are valid

set restart variables to ones from restart image

file_process()

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

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

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

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

item_process()

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

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

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

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

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

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

If the item is a pack item 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 pack item, 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 pack item’s component item to the pack item 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 pack item, and multiplying the pack item’s unit retail. So if the retail contribution of a component item within pack item A is $10, and the same component item’s price_hist record has a retail of $14, and there is only one pack item 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 pack items, always break them down to the pack item level, writing the retail as the pack item multiplied by the component item's price ratio. The pack item itself should never be inserted into the tran_data table.

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

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

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

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

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

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

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

process_detail_error()

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

set_counters()

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

calc_item_totals()

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

calc_prom_totals()

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

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

calc_vendor_funding()

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

process_sales_and_returns()

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

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

process_cancel_records()

If the retail price has changed since the sale occurred, an unjustified loss on the stock ledger vs. the store tables is created. To correct this, this function writes a record to tran_data reversing the price change for the items sold. Either a cancel markup or cancel markdown code is written. The retail amount to be cancelled is the difference between the retail sale price and current retail price multiplied by the total number of items sold or returned.

validate_FHEAD()

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

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

get_eow_eom_date()

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

validate_THEAD()

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

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

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

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

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

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

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

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

get_ref_item()

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

get_item_info()

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

validate_TDETL()

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

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

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

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

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

uom_convert()

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

validate_prom_info()

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

get_deal_contribs()

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

item_store_cursors()

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

new_item_loc()

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

item_check()

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

item_check_orderable()

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

pack_check_orderable()

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

get_vat_rate()

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

pack_check()

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

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

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

item_supplier()

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

get_unit_retail()

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

get_base_price()

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

daily_sales_insert_update()

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

check_daily_exists()

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

process_daily_sales_discount()

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

write_in_store()

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

write_off_retail()

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

remove_stklgdr_vat()

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

write_off_retail_markdowns()

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

write_promotional_markdowns()

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

write_vendor_tran_data()

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

write_wastage_markdown()

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

process_items()

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

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

process_pack()

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

process_packitems()

This function performs processing for the component items of the pack items. 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 pack item 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 pack items retail to come up with the correct retail for each component item. If an employee discount TDETL record has been encountered, an tran_data record with tran_code 60 will be written for each component item.

write_tran_data()

Writes a record to the tran_data insert array.

write_edi_sales()

Writes a record to edi_daily_sales.

update_snapshot()

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

get_454_info()

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

write_vat_err_message()

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

vat_history_data()

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

consignment_data()

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

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

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

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

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

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

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

invc_data()

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

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

find_and_fill_invc_detail ()

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

get_prom_type_info()

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

get_uom_classes()

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

get_item_xform_details()

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

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

get_loc_item_retail()

This function returns the unit_retail from item_loc or item_zone_price.

get_orderable_cost()

This function returns unit_cost from item_supp_country_loc or item_supp_country.

get_orderable_retail()

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

fill_packitem_array()

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

write_item_store_report()

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

posting_and_restart()

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

post_tran_data()

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

post_item_loc()

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

post_item_loc_hist()

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

post_item_loc_hist_mth()

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

post_pack()

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

post_packstore_hist()

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

post_packstore_hist()

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

post_vat_hist_upd()

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

post_ edi_daily_sales_upd ()

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

post_daily_sales_discount ()

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

post_invc_detail_upd ()

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

post_invc_detail_upd ()

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

size_arrays()

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

resize_arrays()

This function reallocates memory for the insert arrays.

write_lock_rej()

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

concession_data()

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

deal_actuals_insert_update ()

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

check_deal_actuals_exists()

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

write_vendor_deal_actuals ()

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

post_deal_actuals ()

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

ON Fatal Error

- Exit Function with -1 return code

ON Non-Fatal Error

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

Input File:

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

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

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

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

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

Record Name	Field Name	Field Type	Default Value	Description
	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
	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

Record Name	Field Name	Field Type	Default Value	Description
	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.
	Sales Sign	Char(1)	‘P’ - positive ‘N’ - negative	Determines if the Total Sales Quantity and Total Sales Value are positive or negative.
	Total Sales Value	Number(20)		Sales value, net sales value of goods sold/returned with 4 implied decimal places.
	Last Modified Date	Char(14)		For VBO future use
	Catchweight Indicator	Char(1)	NULL	Indicates if item is a catchweight item. Valid values are ‘Y’ or NULL
	Actual Weight Quantity	Number(12)	NULL	The actual weight of the item, only populated if catchweight_ind = ‘Y’

Record Name	Field Name	Field Type	Default Value	Description
	Sub Trantype Indicator	Char(1)	NULL	Tran type for ReSA Valid values are ‘A’, ‘D’, NULL
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies transaction record type
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	Promotional Tran Type	Char(6)	promotion type – valid values see code_detail table.	code for promotional type from code_detail, code_type = ‘PRMT’
	Promotion Number	Number(10)	promotion number	promotion number from the RMS
	Sales Quantity	Number(12)		number of units sold in this prom type with 4 implied decimal places.
	Sales Value	Number(20)		value of units sold in this prom type with 4 implied decimal places.
	Discount Value	Number(20)		Value of discount given in this prom type with 4 implied decimal places.
	Promotion Component	Number(10)	NULL	Links the promotion to additional pricing attributes
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type

Record Name	Field Name	Field Type	Default Value	Description
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	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.

Scheduling Considerations

Processing Cycle:	PHASE 2 (daily)
Scheduling Diagram:	This program will likely be run at the beginning of the batch run during the POS polling cycle. It can be scheduled to run multiple times throughout the day, as POS data becomes available.
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

The logical unit of work for the sales/returns upload module will be a valid item sales transaction at a given store location. The location type will be inferred as a store type and the item can be passed as an item or reference item type. The logical unit of work will be defined as a number of these transaction records. The commit_max_ctr field on the restart_control table will determine the number of transactions that equal a logical unit of work.

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

The commit_max_ctr field should be set to prevent excessive rollback space usage, and to reduce the overhead of file I/O. The recommended commit counter setting is 10000 records (subject to change based on experimentation).

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

Pre/Post Functionality for Multi-Threadable Programs [prepost]

Design Overview

The Pre/Post module facilitates multi-threading by allowing general system administration functions (such as table deletions or mass updates) to be completed after all threads of a particular program have been processed. A brief description of all pre- or post-processing functions included in this program can be found in the Function-Level Description section.

This program will take three parameters: username/password to log on to Oracle, a program before or after which this script must run and an indicator telling whether the script is a pre or post function. It will act as a shell script for running all pre-program and post-program updates and purges (the logic was removed from the programs themselves to enable multi-threading & restart/recovery).

For example, to run the pre-program script for the ccext program, the following should be entered on the command line:

```
prepost    user/password    rpl    pre
```

Tables affected:

TABLE	SELECT	INSERT	UPDATE	INDEX	DELETE	TRUNCATE	TRIGGER
all_constraints	Y	N	N	N	N	N	N
all_ind_partitions	Y	N	N	N	N	N	N
all_policies	Y	N	N	N	N	N	N
alloc_detail	Y	N	N	N	N	N	Y
alloc_header	Y	N	N	N	N	N	Y
class	Y	N	N	N	N	N	N
class_sales_forecast	N	N	N	Y	N	Y	N
class_sales_hist	N	N	N	N	Y	N	N
class_sales_hist_mth	Y	N	N	N	Y	N	N
cost_change_trigger_temp	Y	N	N	Y	N	Y	N
cost_susp_head	N	N	Y	N	N	N	N
daily_data_temp	N	N	N	N	N	Y	N
dba_indexes	Y	N	N	N	N	N	N
dba_triggers	Y	N	N	N	N	N	N
dealfct_temp	N	Y	N	N	N	N	N
deal_actuals_forecast	Y	N	N	N	N	N	N
deal_actuals_item_loc	Y	Y	N	N	N	N	N

TABLE	SELECT	INSERT	UPDATE	INDEX	DELETE	TRUNCATE	TRIGGER
deal_bb_no_rebate_temp	N	Y	N	N	N	Y	N
deal_bb_rebate_po_temp	N	Y	N	N	N	Y	N
deal_bb_receipt_sales_temp p	N	Y	N	N	N	Y	N
deal_head	Y	N	N	N	N	N	N
deal_item_loc_explode	Y	N	N	N	N	N	N
deal_sku_temp	N	N	N	Y	N	Y	N
deps	Y	N	N	N	N	N	N
dept_sales_forecast	N	N	N	Y	N	Y	N
dept_sales_hist	N	N	N	N	Y	N	N
dept_sales_hist_mth	Y	N	N	N	Y	N	N
domain_class	N	N	Y	N	N	N	N
domain_dept	N	N	Y	N	N	N	N
domain_subclass	N	N	Y	N	N	N	N
edi_daily_sales	N	N	N	N	Y	N	N
edi_ord_temp	N	N	N	Y	N	Y	N
fif_receiving	N	Y	N	Y	N	Y	N
fixed_deal	Y	N	Y	N	N	N	N
forecast_rebuild	N	N	N	Y	N	Y	N
groups	Y	N	N	N	N	N	N
hist_rebuild_mask	Y	N	N	Y	N	Y	N
ib_results	N	N	Y	N	N	N	N
if_tran_data	Y	N	N	N	N	N	N
invc_detail	N	N	Y	N	N	N	N
invc_detail_temp	Y	N	N	N	N	Y	N
invc_detail_temp2	N	N	N	N	N	Y	N
invc_head	N	N	Y	N	N	N	N
invc_head_temp	Y	N	N	N	N	Y	N
item_forecast	N	N	N	Y	N	N	N
item_loc	Y	N	N	N	N	N	N
item_loc_temp	N	Y	N	N	N	Y	N
item_master	Y	N	N	N	N	N	N

TABLE	SELECT	INSERT	UPDATE	INDEX	DELETE	TRUNCATE	TRIGGER
item_supp_country	Y	N	N	N	N	N	N
item_supp_country_loc	Y	N	N	N	N	N	N
mc_rejections	N	N	N	Y	N	Y	N
mod_order_item_hts	N	N	N	Y	N	Y	N
on_order_temp	N	N	N	Y	N	Y	N
ord_missed	N	N	N	Y	N	Y	N
ord_temp	N	N	N	Y	N	Y	N
ordhead	Y	N	N	N	N	N	N
ordsku	Y	N	N	N	N	N	N
packitem	Y	N	N	N	N	N	N
period	Y	N	N	N	N	N	N
pos_button_head	N	N	Y	N	N	N	N
pos_coupon_head	N	N	Y	N	N	N	N
pos_merch_criteria	N	N	Y	N	N	N	N
pos_mods	N	Y	N	Y	N	Y	N
pos_money_ord_head	N	N	Y	N	N	N	N
pos_payinout_head	N	N	Y	N	N	N	N
pos_prod_rest_head	N	N	Y	N	N	N	N
pos_store	N	N	Y	N	N	N	N
pos_sup_pay_criteria	N	N	Y	N	N	N	N
pos_tender_type_head	N	N	Y	N	N	N	N
reclass_cost_chg_queue	Y	Y	Y	N	N	N	N
reclass_head	Y	N	N	N	N	N	N
reclass_item	Y	N	N	N	N	N	Y
reclass_trigger_temp	Y	N	N	Y	Y	Y	N
repl_attr_update_exclude	Y	N	N	N	Y	N	N
repl_attr_update_head	Y	N	N	N	Y	N	N
repl_attr_update_item	Y	N	N	N	Y	N	N
repl_attr_update_loc	Y	N	N	N	Y	N	N
repl_day	Y	Y	N	N	N	N	N
repl_item_loc	Y	Y	N	N	N	N	N
repl_item_loc_updates	N	Y	N	Y	N	Y	N

TABLE	SELECT	INSERT	UPDATE	INDEX	DELETE	TRUNCATE	TRIGGER
rpl_alloc_in_tmp	N	Y	N	N	N	Y	N
rpl_distro_tmp	N	Y	N	N	N	Y	N
sec_user_zone_matrix	N	N	N	Y	N	Y	N
stage_complex_deal_detail	N	N	N	N	N	Y	N
stage_complex_deal_head	N	N	N	N	N	Y	N
stage_fixed_deal_detail	N	N	N	N	N	Y	N
stage_fixed_deal_head	N	N	N	N	N	Y	N
stake_head	Y	N	N	N	N	N	N
stake_prod_loc	Y	N	N	N	N	N	N
store	Y	N	Y	N	N	N	N
store_add	Y	N	N	N	Y	N	N
subclass_sales_forecast	N	N	N	Y	N	N	N
subclass_sales_hist	N	N	N	N	Y	N	N
subclass_sales_hist_mth	Y	N	N	N	Y	N	N
sup_data	N	N	N	N	Y	N	N
system_options	Y	N	N	N	N	N	N
system_variables	Y	N	Y	N	N	N	N
temp_tran_data	Y	N	N	N	N	Y	N
temp_tran_data_sum	N	Y	N	N	N	Y	N
tif_explode	N	N	N	Y	N	Y	N
tran_data	N	Y	N	N	N	N	N
tsf_head	N	N	Y	N	N	N	N
vat_code_rates	Y	N	N	N	N	N	N
vat_item	Y	N	N	N	N	N	N
week_data_temp	N	N	N	N	N	Y	N
wh	Y	N	N	N	N	N	N
wh_store_assign	N	N	N	N	Y	N	N

Scheduling Constraints

Processing Cycle:	PHASE ALL (daily)
Scheduling Diagram:	See scheduling flow for description of all pre-post requirements in the daily run.
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A (single threaded)

Shared Modules

FORECASTS_SQL.GET_SYSTEM_FORECAST_IND

UDA_SQL.CHECK_REQD_NO_VALUE

FORECASTS_SQL.GET_DOMAIN

ITEM_ATTRIB_SQL.GET_PACK_INDS

FORECASTS_SQL.GET_ITEM_FORECAST_IND

POS_UPDATE_SQL.POS_INVC_DETAIL_INSERT

CAL_TO_454_LDOM

CAL_TO_454_HALF

CAL_TO_CAL_HALF

CAL_TO_CAL_LDOM

CAL_TO_454_WEEKNO

CAL_TO_CAL_WEEKNO

CAL_TO_454

HALF_TO_CAL_FDOH

HALF_TO_CAL_LDOH

HALF_TO_454_FDOH

HALF_TO_454_LDOH

DBMS_RLS.ENABLE_POLICY

Function Level Description

Functions to be used by the individual program functions:

modify_indexes()

This function allows indexes to be disabled or rebuilt before and/or after the action that affects them. The individual program passes in the table name and mode (what action to take “disable” or “rebuild”) and performs that action. The owner of the index is determined using the synonym_trace function in the library oracle.pc.

modify_partition_indexes()

This is called by the modify_indexes function to determine if the indexes that need modified are partitioned indexes. If so, then the statement is modified to take that into account to accomplish the action. Index_owner, index_name and mode is passed to this function. Nothing is passed back out.

truncate_table()

The table_name is passed to this function so that it can be truncated. The owner of the table is determined by using the synonym_trace function in the library oracle.pc.

modify_trigger()

Allows triggers to be disabled or enabled before or after certain processes. The table_name, trigger name and mode(“DISABLE” or “ENABLE”) are passed to this function and the appropriate action is taken. No values are passed back to the calling function.

alter_constraints()

This function diables, enables, or rebuilds a table constraint based on the table name and the mode passed into it. It is called by vendinv_pre().

truncate_user_sec_table()

This is a function used to run the szonrbld pre functions that will truncate the sec_user_zone_matrix table. Disables any indexes prior to the truncation on the associated table and rebuilds/enables them following the truncation.The user running this program for this function must have been granted the ‘drop any table’ and ‘alter any index’ system privilege, or be the owning schema user.

get_454_ldom()

This function calls the procedure CAL_TO_454_LDOM to get the 454 last day of month.

get_454_half()

This function calls the procedure CAL_TO_454_HALF to get the 454 calendar half number.

get_next_454_half()

This function calls the procedure CAL_TO_454_HALF to get the next end-of-month 454 calendar half number.

get_next_cal_half()

This function calls the procedure CAL_TO_CAL_HALF to get the next end-of-month half number on the regular calendar.

get_cal_half()

This function calls the procedure CAL_TO_CAL_HALF to get the half number on the regular calendar

get_cal_ldom()

This function calls the procedure CAL_TO_CAL_LDOM to get the end of the month on the regular calendar.

get_454_weekno()

This function calls the procedure CAL_TO_454_WEEKNO to get the 454 week number in half.

get_cal_weekno()

This function calls the procedure CAL_TO_CAL_WEEKNO to get the week number in half on the regular calendar.

get_454_date()

This function calls the procedure CAL_TO_454 to get the 454 calendar week number.

get_cal_fdoh()

This function calls the procedure HALF_TO_CAL_FDOH to get the first day of half.

get_cal_ldoh()

This function calls the procedure HALF_TO_CAL_LDOH to get the last day of half.

get_454_fdoh(void);

This function calls the procedure TO_454_FDOH to get the first day of half in 454 calendar.

get_454_ldoh(void)

This function calls the procedure HALF_TO_454_LDOH to get the last day of half in 454 calendar.

get_tomorrow()

This function gets the next day after the vdate.

get_forecast_ind()

This function cals FORECASTS_SQL.GET_SYSTEM_FORECAST_IND to get the system_forecast_ind.

validate_reclassify()

Validates the reclassification. If the reclassification is rejected, then the data from the RECLASS_TRIGGER_TEMP table is deleted, else the data is inserted into RECLASS_COST_CHG_QUEUE table.

check_stock_count()

This function checks for the existence of a stock count of an item in the STAKE_SKU_LOC or STAKE_PROD_LOC.

check_order()

This function checks for the existence of an order for an item in the ORDHEAD and ORDSKU tables.

check_uda()

This function calls UDA_SQL.CHECK_REQD_NO_VALUE which determines if an item's new hierarchy has any required UDA defaults that the item is not currently associated with.

check_domain_exists()

This function calls FORECASTS_SQL.GET_DOMAIN to check for the existence of the domain for a merchandise hierarchy.

check_forecast()

This function validates the reclassification of an item based on forecast indicator. First, it checks if the item passed is a pack through the package call to ITEM_ATTRIB_SQL.GET_PACK_INDS. Then for non-pack items, it calls FORECASTS_SQL.GET_ITEM_FORECAST_IND to get the item forecast indicator.

delete_reclass_trigger_temp()

This function deletes the records for a given item from the RECLASS_TRIGGER_TEMP.

Individual Program Functions

rpl_pre()

This function truncates the following tables before replenishment extracts are performed:

- ORD_TEMP
- ORD_MISSED

It also disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the ‘drop any table’ and ‘alter any index’ system privilege, or be the owning schema user.

salweek_post()

Updates the last end-of-week date on the SYSTEM_VARIABLES table to the run date after all weekly stock ledger data has been processed.

salmth_post()

Updates the following SYSTEM_VARIABLES columns to reflect the current date’s values after all monthly stock ledger data has been processed:

- last_eom_half_no
- last_eom_month_no
- last_eom_date
- next_eom_date
- last_eom_start_half
- last_eom_end_half
- last_eom_start_month
- last_eom_mid_month
- last_eom_next_half_no

- last_eom_day
- last_eom_week
- last_eom_month
- last_eom_year
- last_eom_week_in_half

rplatupd_pre()

This function truncates the MC_REJECTIONS table so that it is free to hold new mass change rejections. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the ‘drop any table’ and ‘alter any index’ system privilege, or be the owning schema user.

rplatupd_post()

This function truncates the holding tables REPL_ATTR_UPDATE_ITEM and REPL_ATTR_UPDATE_LOC after their records have been processed. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the ‘drop any table’ and ‘alter any index’ system privilege, or be the owning schema user.

rilmaint_post()

This function truncates the REPL_ITEM_LOC_UPDATES table after these records are processed so the table is free to hold new updates. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the ‘drop any table’ and ‘alter any index’ system privilege, or be the owning schema user.

supmth_post()

Deletes records from table SUP_DATA after all daily supplier data records have been rolled up to month level.

sccext_post()

Updates all processed supplier cost change record status to ‘Extracted’.

hstbld_pre()

Deletes sales history data for the dept exists in the table hist_rebuild_mask from the three tables subclass_sales_hist, class_sales_hist and dept_sales_hist prior to running hstbld in rebuild mode.

hstbld_post()

This function truncates the holding table MASK_REBUILD after building history records. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the ‘drop any table’ and ‘alter any index’ system privilege, or be the owning schema user.

posdnld_post()

This clears the POS_MODS table after all records have been downloaded to the POS. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the ‘drop any table’ and ‘alter any index’ system privilege, or be the owning schema user.

poscdnld_post()

This clears the config_status and loc_grp_status in POS_LOC_GRP and sets all values of extract_req_ind to ‘N’. It clears the status column in POS_MERCH_CRITERIA. It also sets the status_ind column in POS_STORE to ‘N’.

reqext_post()

This function updates the TSFHEAD table and sets the status to ‘A’, approval_id to ‘BATCH’, approval_date to the vdate, and the repl_tsf_approve_ind to ‘N’ where the repl_tsf_approve_ind is equal to ‘Y’.

likestore_post()

This function should only be run after both storeadd.pc and all threads of likestore.pc have successfully completed.

In the REPL_ITEM_LOC, table, likestore_post selects and inserts all information from the a like store for the new store.

stkupd_pre()

Calls the stored function DBMS_MVIEW.REFRESH.

stkupd_post()

This function disables the RMS_COL_ITL_UR_AUR trigger of ITEM_LOC.

dtesys_post()

Enables the RMS_COL_ITL_UR_AUR trigger of ITEM_LOC table.

ociroq_pre()

This function truncates the rpl_net_inventory_tmp table, which is populated by the ociroq.c and queried from reqext.pc. This function also inserts records into RPL_DISTRO_TMP values from ALLOC_DETAIL, and ALLOC_HEAD table, and into RPL_ALLOC_IN_TMP values from ALLOC_DETAIL, ALLOC_HEAD, and ORDHEAD table. This function also creates a unique index in these two destination tables.

rplext_post()

Truncates the tables RPL_DISTRO_TMP, and RPL_ALLOC_IN_TMP.

posupld_post()

This updates the columns total_merch_cost , total_qty, invc_qty, INVC_HEAD tables based on the corresponding columns in the INVC_HEAD_TEMP table.

vatdlxpl_post()

This inserts into pos_mods all transaction level items on the vat_item table where the item has a new tran_code. Also, if a sub-transaction level item is on vat_item, it is inserted into the pos_mods table, along with its parent item. These items are not picked up by the vatdlxpl program because the vat_code rate has not changed.

saleoh_pre()

Calculates the next_eom_date, and updates the SYSTEM_VARIABLES.

dealday_pre()

This gets the total sales and purchases from the TEMP_TRAN_DATA table and inserts a new record in TEMP_TRAN_DATA_SUM based on dept, class, subclass, loc_type, location, tran_date, and tran_code.

dealday_post()

Copies the contents of the table TEMP_TRAN_DATA_SUM into TRAN_DATA table. Afterwards, then TEMP_TRAN_DATA_SUM is truncated.

hstbldmth_post()

This is responsible for deleting records in the following tables:

- CLASS_SALES_HIST_MTH
- SUBCLASS_SALES_HIST_MTH
- CLASS_SALES_HIST_MTH
- DEPT_SALES_HIST_MTH

The following functions should be run after the edidlprd program.

edidlprd_post()

Deletes old records from the EDI_DAILY_SALES table after they have been processed.

fcstrbld_post()

This truncates the holding table FORECAST_REBUILD after all records have been processed. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the ‘drop any table’ and ‘alter any index’ system privilege, or be the owning schema user.

vrplbld_post()

This truncates the EDI_ORD_TEMP table after all replenishment orders have been build from the data held there. Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the ‘drop any table’ and ‘alter any index’ system privilege, or be the owning schema user.

cntrordb_post()

Sets the last_cont_order_date on system_variables to vdate.

fifgldn1_post()

If Oracle Financials is being used, delete everything from the fif_receiving table and repopulate it from the if_tran_data table. Disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the ‘drop any table’ and ‘alter any index’ system privilege, or be the owning schema user.

fsadnld_post()

Updates the load_sales_ind to ‘N’ for all records on the appropriate domain table – domain_dept, domain_class, or domain_subclass, where system_options.domain_level = ‘D’, ‘C’, or ‘S’, respectively.

policy_enable()

Enables or disables policies.

whstrasg_post ()

Deletes all warehouse store assignment records from the warehouse store assignment table if the assignment date (wh_store_assign.assign_date) is less than or equal to the current date (period.vdate) minus the warehouse store assignment history days (system_options.wh_store_assign_hist_days).

costcalc_post()

This truncates the deal_sku_temp table. This disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the ‘drop any table’ and ‘alter any index’ system privilege, or be the owning schema user.

tifposdn_post()

This truncates tif_explode table. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the ‘drop any table’ and ‘alter any index’ system privilege, or be the owning schema user. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation.

htsupld_pre()

This truncates the mod_order_item_hts table so that reports will be correct and not include data from previous runs of htsupld. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the ‘drop any table’ and ‘alter any index’ system privilege, or be the owning schema user.

onordext_pre()

This truncates the on_order_temp table. It disables any indexes prior to the truncation on the associated tables and rebuilds/enables them following the truncation. The user running this program for this function must have been granted the ‘drop any table’ and ‘alter any index’ system privilege, or be the owning schema user.

precostcalc_pre()

This processeses records from the COST_CHANGE_TRIGGER_TEMP and RECLASS_TRIGGER_TEMP tables. Reclass_trigger_temp is populated only by database trigger and cost_change_trigger_temp is populated by database trigger and edi_cost_change_sql.create_cost_chg.

This function will either insert new records or update existing ones on reclass_cost_chg_queue. Both tables, COST_CHANGE_TRIGGER_TEMP and RECLASS_TRIGGER_TEMP are truncated and their indexes rebuilt at the end of this function. The user running this program for this function must have been granted the ‘drop any table’ and ‘alter any index’ system privilege, or be the owning schema user.

reclsdly_pre()

This disables the trigger RMS_TABLE_RCS_BIDR on the reclass_item table. The user running this program for this function must have been granted the ‘alter any trigger’ system privilege, or be the owning schema user.

ibcalc_pre()

This updates the status on ib_results to ‘U’nprocessed where the status = ‘W’orksheet so after ibcalc is run, multiple records in ‘W’orksheet status will not exist for each item/location.

fcstprg_pre()

This disables any indexes prior to the truncation on following tables. This is run BEFORE the fcstprg.pc program on PARTITIONED TABLES only:

- ITEM_FORECAST
- DEPT_SALES_FORECAST
- CLASS_SALES_FORECAST
- SUBCLASS_SALES_FORECAST

The user running this program for this function must have been granted the ‘alter any index’ system privilege, or be the owning schema user.

fcstprg_post()

This rebuilds the indexes following truncation of following tables:

- ITEM_FORECAST
- DEPT_SALES_FORECAST
- CLASS_SALES_FORECAST
- SUBCLASS_SALES_FORECAST

The user running this program for this function must have been granted the ‘alter any index’ system privilege, or be the owning schema user.

dealinc_pre()

Call get_sys_date()

Call size_arrays()

Loops through the deal actuals item loc table and create any item/loc/order combinations in the table that have previous turnovers but do not exist in future periods.

dealfct_pre()

This inserts details of forecast periods for active deal components that require processing into dealfct_temp table.

dealact_pre_no_rebate()

Truncates the deal_bb_no_rebate_temp table.

Then inserts billback NO Rebate type of deal into deal_bb_no_rebate_temp.

dealact_pre_rebate_po()

Truncates the deal_bb_rebate_po_temp table.

Then inserts billback rebate PO type of deal into deal_bb_rebate_po_temp.

dealact_pre_receipt_sales ()

Truncates the deal_bb_receipt_sales_temp.

Then inserts billback rebate Sales and Receipt type of deal into deal_bb_receipt_sales_temp.

vendinvc_pre()

Truncate the STAGE_COMPLEX_DEAL_HEAD table.

Truncate the STAGE_COMPLEX_DEAL_DETAIL table.

Then inserts complex deals for invoicing into vendinvc_temp.

vendinvf_pre()

Truncate the STAGE_FIXED_DEAL_HEAD table.

Truncate the STAGE_FIXED_DEAL_DETAIL table.

vendinvc_post()

Get vdate.

Call process_deal_head().

vendinvf_post()

Get vdate.

Call process_fixed_deal().

process_fixed_deal()

For each active Fixed Deal record where the Collect End Date is earlier than the vdate, set it's status to Inactive.

process_deal_head()

For each active Deal Head record where Est Next Invoice Date, Close Date, Last Invoice Date and Last EOM Date are earlier than vdate, AND Billing Type is Off Invoice and Invoice processing Logic !='NO', set the Est Next Invoice Date to null.

Item requisition extraction [reqext]

Design Overview

Reqext (Item Requisition Extraction) handles automatic replenishment of items from warehouses to stores. It cycles through every item-store 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 store (if any). In addition, it distributes this Recommended Order Quantity (ROQ) over any applicable alternate items associated with the item. The program then takes this information and either creates new transfer line items or adds to existing ones.

Alternate items are either simple packs or substitute items. Simple packs are sellable and orderable packs that contain only a single item, such as a six-pack of cola or twelve-pack of socks. Substitute items are items predefined to be interchangeable with the item being replenished (referred to as the master item).

When an item is set up to use simple packs (designated by an indicator on the REPL_ITEM_LOC table), the ROQ must be distributed among these packs according to desirability. If a master item has no simple packs associated with it, it will be requested as itself. If there is only one pack associated with the item (referred to as the primary simple pack), then there is no distribution needed – the item will be transferred in this simple pack, since the cost per item for a pack is always less than that of an individual item. If multiple simple packs can be substituted for an item, then the distribution of the ROQ over these packs is determined by comparing the packs' relative sales history. Replenishing an item through multiple simple packs can have a severely negative effect on the performance of this program! Because the pack distribution depends on access to the huge sales history tables (ITEM_LOC_HIST), it is not recommended that many items be placed on replenishment through multiple simple packs. Whenever possible, it is better to assign a primary simple pack to the item, since this does not require distribution calculation.

If an item is not set up to use simple packs, the program will see if any substitute items are associated with it. If there are no substitute items associated with the master item, it will be transferred alone. If there are substitute items, they will be fetched into a list and the master item placed at either the head or tail end of the list, depending on the fill priority (set on the SUB_ITEMS_HEAD table). The priority determines which items are transferred first.

No matter what type of alternate items (if any) are used, the program will account for availability when building transfer line items. For simple packs, the share of ROQ allocated to each pack may be decreased or increased if the source warehouse has a shortage of some packs but a surplus of others. For substitute items, transfer quantities are prorated by calculating the ratio of total availability to total need, and items are transferred in order of priority until all need is filled or until no stock is available.

Once the transfer quantity of an item has been calculated, the transfer line item is posted to the database if 1) the actual quantity to transfer is greater than zero, and 2) the replenishment order control indicator for the item-store combination is either Automatic or Semi-Automatic. If it is Manual, a record will be written to another table (REPL_RESULTS) for reporting purposes. If the system-level All Replenishment Results indicator is set to "Yes", all line items will be written to REPL_RESULTS, even if the quantity to order is zero. Whenever a transfer line item is placed, the appropriate item-location table (ITEM_LOC_SOH) is updated to reflect the fact that stock is now reserved for transfer at the warehouse and expected at the store.

Tables Affected:

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_LOC	Yes	No	No	No
ITEM_LOC_SOH	No	No	Yes	No
ITEM_MASTER	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
PACKHEAD	Yes	No	No	No
PACKITEM	Yes	No	No	No
PACKSTORE_HIST	Yes	No	No	No
PERIOD	Yes	No	No	No
RAG_SKUS_ST_HIST	Yes	No	No	No
REPL_DAY	Yes	No	No	No
REPL_ITEM_LOC	Yes	No	Yes	No
REPL_RESULTS	No	Yes	No	No
RPL_NET_INVENTORY_TMP	Yes	No	No	No
STORE	Yes	No	No	No
SUB_ITEMS_DETAIL	Yes	No	No	No
SUB_ITEMS_HEAD	Yes	No	No	No
SUPS	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No
TSFDETAIL	Yes	Yes	Yes	No
TSFHEAD	Yes	Yes	No	No
WH	Yes	No	No	No

Scheduling Constraints

Processing Cycle:	PHASE 3
Scheduling Diagram:	Rplatupd, repladj, prepost ociroq and ociroq need to run before reqext so that all replenishment calculation attributes are up to date. Posupld need to run before reqext so that all stock information is up to date. Rplext should run after reqext, since the ROQ for a warehouse is influenced by any transfers created.
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	ITEM

The restart_control.num_threads for the batch program reqext.pc should be equal to the number of partitions on the rpl_net_inventory_tmp table.

Restart Recovery

The logical unit of work is item, source warehouse. The driving cursor is ordered by item, source warehouse, order control indicator and simple pack indicator. When any of these values change during the course of processing (i.e., the current value is different than that of the previous record), then a transfer will be created, taking total quantities and availability into consideration (see replenish_item(), below).

Shared Modules

ITEM_ATTRIB_SQL.GET_ITEM_MASTER: Stored PL/SQL procedure for getting a record containing every attribute from ITEM_MASTER for the entered item.

TRANSFER_SQL.AUTOMATIC_BT_EXECUTE: Stored PL/SQL procedure that will automatically handle inventory updates and close out a transfer by calling two other functions, TRANSFER_OUT_SQL.EXECUTE_BT, and TRANSFER_IN_SQL.EXECUTE_BT.

TRANSFER_SQL.NEXT_TSF_PO_LINK_NO: Stored PL/SQL procedure that generates the next tsf_po_link sequence number.

ITEMLOC_QUANTITY_SQL.GET_LOC_CURRENT_AVAIL: Stored PL/SQL procedure for calculating the current available stock of a given item available at a given location.

NEXT_TRANSFER_NUMBER: Stored PL/SQL procedure used for getting the next valid transfer number for use in creating new transfers.

Function Level Description

main()

The standard Retek main function, this calls init(), process() and final(), and posts messages to the daily log files.

init()

Initializes the Restart-Recovery API and fetches system-level global variables.

driving_cursor()

Opens, fetches data from, or closes the driving cursor. The driving cursor uses dynamic SQL that accepts the partition number from the rpl_net_inventory_tmp table as an argument. This is a support function for process().

process()

This function fetches records from the driving cursor (driving_cursor()), passes them to replenish_item() to perform all appropriate actions, and commits work when appropriate (post_all(), restart_commit()).

replenish_item()

The controlling function for replenishment calculations. This function copies records out of the driving cursor buffer (copy_repl_to_store()). If a change in item, source warehouse, order control indicator, or simple pack indicator has occurred, the appropriate functions are called to calculate distribution of need over all appropriate alternate items and stores, and to place the transfers. If item's ROQ is zero or negative, no mater simple pack indicator is on the master item will be used for replenishment (build_pack_ratio(), calc_pack_dist(), calc_sub_dist()).

place_tsf_line_item()

This function takes a item-location combination and a transfer quantity, and actually builds the transfer line item (handle_tsf()). It then updates the item-location tables to reflect the change in stock (handle_item_loc()), and writes a record to the reporting table (handle_repl_results()) when appropriate.

final()

The standard Retek final function, this closes down the Restart-Recovery API.

Substitute Item Distribution and Transfer

calc_sub_dist()

Calculate distribution of the ROQ over substitute items. Substitute items are items (selected by the user beforehand) that can be requested in place of a given item to cover situations where availability is too low or demand is too high.

If the total ROQ for the item is equal or less than zero then only the master item is used. Call out function add_master_item(). If stock category is anything other than WH/Cross Link, call out get_sub_items else if the stock category is WH/Cross Link, substitute items won't be supported. Instead, another path will be provided in the program to add in the master item and retrieve its availability, which is to call function get_mbr_items.

After calling get_sub_items() to generate a list of appropriate items for transfer, the function loops through every item-location combination and performs the following steps to make sure that both need and availability are accounted for when placing transfers from the warehouse to the stores:

- If the total availability of all items in the substitute list cannot cover the full need over all stores, then the ratio of the total availability to the total ROQ is calculated. If total availability can cover total ROQ, the ratio is set to 1.
- The initial transfer quantity for the item at the location is calculated as the store’s need adjusted by the availability ratio, and rounded up to a receivable pack size.
- If there is not enough of the item available at the warehouse to fill the calculated transfer quantity, the quantity will be decremented to an orderable amount.
- The transfer line item is placed by calling place_tsf_line_item().
- The store’s ROQ, total ROQ, availability of the item, and total availability are all decremented by the amount just transferred to prepare for the next item-location’s calculation.

get_sub_items()

Retrieves substitute items for the current master item and information about them from the database (receiving pack size, availability, etc.). If the fill priority for this set of items (SUB_ITEMS_HEAD.fill_priority) is set to ‘M’aster, the master item will be the first one in the list, and will be used first to fill need. If it is set to ‘S’ubstitute, the master item will be placed at the tail end of the list. This is a support function calc_sub_dist().

get_mbr_items()

Retrieves the master item and its source wh availability in order to process warehouse docked/crossdocked transfers for market based replenishment. Because this new stock category doesn't currently support the use of substitute items, this simple master item retrieval path is being provided. This is a support function for calc_sub_dist().

add_master_item()

Adds the master item to the appropriate position in the substitutes list. This is a support function for get_sub_items().

shift_subs()

If the fill priority for the substitutes list is set to 'M'aster, the master item must be placed at the head of the list. This function clears out the first position by moving each substitute item 'back' a slot. This is a support function for get_sub_items().

get_item_loc_info()

Retrieves a store order multiple and rounding information for every item-store combination. Also, retrieve the status of the substitution item at the store it is potentially being transferred to. If the item is Inactive at the store location, then the transfer won't be created for that item. This is a support function for calc_sub_dist().

Database DML Handling

post_all()

The DML handling functions (handle_tsf(), handle_item_loc(), handle_repl_results()) normally only post information to the database tables when their respective buffers are full. When a commit point is reached, however, all buffers must be flushed to ensure restartability. This function forces all the buffers to be posted to the database.

handle_tsf()

Controls handling of inserts and updates to the Transfer tables.

add_tsfhead()

Deals with transfer header information. Either finds an appropriate transfer to add line items to (matching to/from locations, department and freight code), or creates a new one. passes back the transfer number for use in add_tsfdetail(). This is a supporting function for handle_tsf().

add_tsfdetail()

Deals with transfer detail information. Either finds an appropriate record on the TSFDETAIL table to add quantity to (matching transfer number and item), or creates a new one if none is found. This is a supporting function for handle_tsf().

get_next_seq_no()

Every line item on a transfer has a unique identifier within that transfer. This function gets the next sequence number for a new line item. This is a supporting function for add_tsfdetail().

post_tsf()

Posts transfer information to the database. Inserts to TSFHEAD, inserts and updates to TSFDETAIL. This is a supporting function for handle_tsf().

handle_item_loc()

Whenever a transfer is created or modified, the source location's reserved quantity and the receiving location's expected quantity must be adjusted to reflect the new stock status. This function controls the handling of updates to the item-location tables add_item_loc()

Adds records to arrays for update of expected and reserved quantities on the item-location table (ITEM_LOC_SOH) based on the appropriate item types. This is a support function for handle_item_loc().

post_item_loc()

Posts item-location stock status changes to the database (ITEM_LOC_SOH). This is a support function for handle_item_loc().

handle_repl_item_loc()

Controlling function for handling updates to the REPL_ITEM_LOC table.

add_repl_item_loc()

This function adds records to the repl_item_loc array for the update of the REPL_ITEM_LOC table.

post_repl_item_loc()

This function posts repl_item_loc records to the database. Also updates the the last_review_date and last_delivery_date columns on REPL_ITEM_LOC to reflect the fact that item-location combinations have just been evaluated.

handle_repl_results()

Controls posting of report information to the REPL_RESULTS table.

add_repl_results()

Adds records to the replenishment results structure for reporting. This is a supporting function for handle_repl_results().

post_repl_results()

Posts replenishment information to the REPL_RESULTS table. This is a supporting function for handle_repl_results().

PL/SQL Stored Procedure Calls

handle_wh_current_avail()

Gets the available quantity of a given item at a given warehouse. Calls out get_wh_current_avail() and create_book_transfer()

create_book_transfer()

Creates transfers from one VWH to another when one VWH in a physical WH fails to have enough available inventory to satisfy a store's ROQ for a particular item. This function is also a wrapper for TRANSFER_SQL.AUTOMATIC_BT_EXECUTE stored PL/SQL procedure

get_wh_current_avail()

Gets the available quantity of a given item at a given warehouse. This function is a wrapper for the ITEMLOC_QUANTITY_SQL.GET_LOC_CURRENT_AVAIL stored PL/SQL procedure.

next_transfer_number()

Gets the next transfer number in the Oracle stored sequence for creating new transfer headers. This function is a wrapper for the NEXT_TRANSFER_NUMBER stored procedure.

next_tsf_po_link_no()

Gets the next tsf_po_link_no in the stored sequence for creating market based warehouse stocked/cross-docked transfers. This function is a wrapper for the TRANSFER_SQL.NEXT_TSF_PO_LINK_NO stored PL/SQL procedure.

Domain Validation

Support Functions

copy_repl_to_store()

Copies a record from the structure holding rows from the driving cursor into a structure holding item-location information for ROQ calculation, distribution, and transfer placement.

reset_store_struct()

Resets summary variables in a store information structure to prepare it for the next set of line items.

reset_alt_item_struct()

Resets summary variables in an alternate item structure to prepare it for the next set of alternates.

Array Sizing

size_repl_info_struct()

Allocates memory to the structure used to buffer fetches from the driving cursor.

size_store_struct()

Allocates memory to the structure used to hold item-location level information.

size_alt_item_struct()

Allocates memory to the structure used to hold information about alternate items (either simple packs or substitute items).

size_tsfhead_struct()

Allocates memory to the structure used to buffer inserts to the Transfer Header table.

size_tsfdetail_struct()

Allocates memory to structures used to buffer inserts and updates to the Transfer Detail table.

size_item_loc_struct()

Allocates memory to structures used to buffer updates of the item-location tables

size_repl_results_struct()

Allocates memory to the structure used to buffer inserts to the Replenishment Results table.

size_book_transfer_struct()

Allocates memory for the elements of a item_loc insert structure.

size_repl_update_struct()

Allocates memory for the elements of a repl_item_loc update structure.

Monthly Replenishment Purge [rplprg_month]

Design Overview

The replenishment extraction programs (rplext, reqext) write a number of records to REPL_RESULTS, the investment buy process writes records to IB_RESULTS and the Buyer Worksheet Form populates BUYER_WKSHT_MANUAL. These tables hold information that is relevant to replenishment processes. Over time, records on these tables become unneeded and should be cleared out. The Monthly Replenishment Purge Program goes through these tables and clears out those records that are older than a predetermined number of days and drops the partitions created in the REPL_RESULTS table. The eways ewInvAdjustToRMS, ewRecieptToRMS need to be shutdown when rplprg_month.pc is run..

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
ALL_TAB_PARTITIONS	No	Yes	No	No	No
REPL_RESULTS	No	No	No	No	Yes
BUYER_WKSHT_MANUAL	No	No	No	No	Yes
STORE_ORDERS	No	No	No	No	Yes
IB_RESULTS	No	No	No	No	Yes

Function Level Description

main()

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

init()

This function fetches various threshold dates. These dates are the current processing date (PERIOD.vdate) minus SYSTEM_OPTIONS.repl_results_purge_days, PERIOD.vdate minus SYSTEM_OPTIONS.store_orders_purge_days and SYSTEM_OPTIONS.ib_results_purge_days. This function should also fetch the commit_max_ctr from RESTART_CONTROL to determine how many records can be deleted between commits

process()

This program will, in four different loops, delete from REPL_RESULTS, BUYER_WKSHT_MANUAL, STORE_ORDERS and IB_RESULTS where the repl_date, create_date, need_date and create_date are less than their associated threshold date and the rownum is less than the commit counter. The BUYER_WKSHT_MANUAL table will be purged using the replenishment results purge days threshold date. If REPL_RESULTS is partitioned, this program will drop the partitions that were created and then validate all the invalid objects of the schema owner.

If this statement processes no rows, then the table has been fully purged and the function should return successfully. Otherwise, it should commit and return to the top of the loop. The loop with intermittent commits is necessary in order to ensure that rollback segments are not in danger of overflowing.

final()

This function will terminate restart-recovery. It also calls retek_close() to refresh the current thread.

Scheduling Considerations

Processing Cycle:	Phase 3 (monthly).
Scheduling Diagram:	Can run anytime.
Pre-Processing:	N/A.
Post-Processing:	N/A.
Threading Scheme:	N/A

Restart Recovery

Logical Unit of Work (recommended Commit check points)

Driving Cursor

Because this program performs only deletes, there is no need for restart/recovery or multithreading, and there is no driving cursor. However, this program still needs an entry on RESTART_CONTROL to determine the number of records to be deleted between commits.

Technical Issues

Think about using Parallel Query in the delete.

If the client doesn't want to hold on to their results, it's probably better to remove this program from the schedule and modify rpl_pre to truncate REPL_RESULTS nightly (similar to the current treatment of ORD_SUGG/F).

Testing Scenarios

Insert records onto REPL_RESULTS, BUYER_WKSHT_MANUAL, STORE_ORDERS and IB_RESULTS some with a repl_date, create_date or need_date earlier than the threshold date, some later. Run rplprg_month. Those records earlier than the threshold should be deleted, while those later should remain.

Process enough records (or set the commit_max_ctr low enough) so that the program must perform multiple deletes in process ().

Sales Audit Get Reference [sagetref]

Design Overview

This program will fetch all reference information needed by saimplog.pc and write this information out to separate output files. One file will contain a listing of all items in the system. A second file will contain information about all items that have wastage associated with them. A third file will contain reference items. A fourth file will contain primary variant information. A fifth file will contain all variable weight UPC definitions in the system. A sixth file will contain all of the valid store/day combinations in the system. A seventh file will contain all code types and codes used in field level validation. An eighth file will contain all error codes, error descriptions and systems affected by the error. A ninth file will contain the credit card validation mappings. A tenth file will contain the store_pos mappings. An eleventh file will contain the tender type mappings. A twelfth file will contain the merchant code mappings. A thirteenth file will contain the partner mappings. A fourteenth file will contain the supplier mappings. A fifteenth file will contain employee mappings. Finally a sixteenth file will contain banner information. These files will be used by the automated audit to validate information without repeatedly hitting the database.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ITEM_MASTER	No	Yes	No	No	No
ITEM_LOC	No	Yes	No	No	No
VAR_UPC_EAN	No	Yes	No	No	No
SA_IMPORT_LOG	No	Yes	No	No	No
CURRENCIES	No	Yes	No	No	No
STORE_DAY	No	Yes	No	No	No
STORE	No	Yes	No	No	No
SA_STORE_DAY	No	Yes	No	No	No
CODE_DETAIL	No	Yes	No	No	No
SA_ERROR_CODES	No	Yes	No	No	No
SA_CC_VAL	No	Yes	No	No	No
SA_STORE_POS	No	Yes	No	No	No
POS_TENDER_TYPE_HEAD	No	Yes	No	No	No
NON_MERCH_CODE_HEAD	No	Yes	No	No	No
PARTNER	No	Yes	No	No	No
SUPS	No	Yes	No	No	No
SA_STORE_EMP	No	Yes	No	No	No
BANNER	No	Yes	No	No	No

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
CHANNELS	No	Yes	No	No	No
ADDR	No	Yes	No	No	No

Function Level Description

main()

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

init()

This function will initialize the necessary restart recovery variables.

Calls the function retek_init().

process()

This will call process_item_master_info() to retrieve item information from the database and write it to a item and waste output file. This function will then call process_ref_item_info() to retrieve reference item information from the database and write it to the reference item output file. This function will also call process_prim_variant_info() to retrieve primary variant information from the database and write it to a primary variant output file. This function will then call process_var_upc_ean_info() to retrieve all variable weight UPC mappings from the database and write them to the variable weight UPC output file. This function will then call process_store_day_info() to retrieve all valid store day combinations from the database and write them to the store day output file. This function will then call process_codes_info() to retrieve all codes from the database that are used in file validation and write them to the codes output file. This function will then call process_error_info() to retrieve all errors from the database that are used in file validation and write them to the error output file. This function will then call process_cc_info() to retrieve all credit card validation mappings from the database and write them to the credit card validation output file. This function will then call process_store_pos_info() to retrieve all store/pos mappings from the database and write them to the store POS output file. This function will then call process_tender_type_info() to retrieve all tender type mappings from the database and write them to the tender type output file. This function will then call process_merch_codes_info() to retrieve all merchant code mappings from the database and write them to the merchant code output file. This function will then call process_partner_info() to retrieve all partner mappings from the database and write them to the partner output file. This function will then call process_supplier_info() to retrieve all supplier mappings from the database and write them to the supplier output file. This function will then call process_employee_info() to retrieve all employee mappings from the database and write them to the employee output file. Finally this function will call process_banner_info() to retrieve banner information from the database and write them to the banner output file.

process_item_master_info()

This function will query information for all sellable items from the item_master table and uses this information to populate the item master array. This includes all items (whose item status = ‘A’ and tran_level = item_level and sellable_ind = ‘Y’). This function also calls size_item_master_arrays() to allocate memory for the item master array. The columns that are selected for this process include item, dept, class, subclass, waste_type, waste_pct, standard_uom, and catch_weight_ind. The information is ordered by item. All records in the item master array should be written to the item data output file by calling write_item_data(). Only records in which waste_type or waste_pct are not null should be written to the waste data file by calling write_waste_data().

size_item_master_arrays()

This function allocates memory for the item master array used in process_item_master_info.

write_item_data()

This function will write all elements of the item master array to the item data output file. The file format for the item data file can be found in the I/O section of this document. The information should be ordered by item.

write_waste_data()

This function will accept the entire item master array as input, but will only write records to the waste data file if the waste_type or waste_pct for the item are not null. This function then checks to make sure that data that came back as NULL is actually blank. The file format for the waste data file can be found in the I/O section of this document. The information should be ordered by item.

process_ref_item_info()

This function will query item reference information for all sellable items from the item_master table and uses this information to populate the ref item array. This includes all items (whose item status = ‘A’ and item_level – tran_level = 1 and sellable_ind = ‘Y’). This function also calls size_ref_item_arrays() to allocate memory for the ref item array. The columns that are selected for this process include item and item_parent. The information is ordered by item. All records in the ref item array should be written to the ref item data out put file by calling write_ref_item_data().

size_ref_item_arrays()

This function allocates memory for the ref item array used in process_ref_item_info.

write_ref_item_data()

This function will write all elements of the ref item array. The file format for the ref item data file can be found in the I/O section of this document. The information should be ordered by item.

process_prim_variant_info()

This function will query primary variant information for all items from the item loc and item_master tables and uses this information to populate the primary variant array. This includes all items (whose item status = ‘A’ and item_level – tran_level = 1 and primary_variant is NOT NULL). This function also calls size_prim_variant_arrays() to allocate memory for the primary variant array. The columns that are selected for this process include loc, item and primary variant. The information is ordered by loc (alphabetically not numerically) and then by item. All records in the primary variant array should be written to the primary variant data out put file by calling write_prim_variant_data().

size_prim_variant_arrays()

This function allocates memory for the primary variant array used in process_prim_variant_info.

write_prim_variant_data()

This function will write all elements of the prime variant array. The file format for the primary variant data file can be found in the I/O section of this document. The information should be ordered by loc (alphabetically not numerically) and then by item.

process_var_upc_ean_info()

This function will query variable weight UPC information from the var_upc_ean and item_master tables and uses this information to populate the variable weight UPC array. This includes all distinct var_upc_ean records whose format_id = item_master.format_id and item status = ‘A’. This function also calls size_var_upc_ean_arrays() to allocate memory for the variable weight UPC array. The columns that are selected for this process include format_id, format_desc, prefix_length, begin_item_digit, begin_var_digit, check_digit, default_prefix and prefix. The information is ordered by format_id. All records in the variable weight UPC array should be written to the variable weight UPC output file by calling write_var_upc_info()

size_var_upc_ean_arrays()

This function allocates memory for the variable weight UPC array used in process_var_upc_ean_info.

write_var_upc_data()

This function will write all elements of the variable weight UPC array. The file format for the variable weight UPC file can be found in the I/O section of this document. The information should be ordered format_id.

process_store_day_info()

This function will query all valid store/day combinations from the sa_store_day, store, sa_import_log and currencies tables and uses this information to populate the store day array. This includes all stores which sa_import_log.system_code = ‘POS’. This function also calls size_store_day_arrays() to allocate memory for the store day array. The columns that are selected for this process include store, business_date, store_day_seq_no, day, tran_no_generated, decode system_code (code = ‘POS’) and currency_rtl_desc. The information should be ordered by store (alphabetically not numerically) and business date. All records in the store day array should be written to the store day output file by calling write_store_day_data().

size_store_day_arrays()

This function allocates memory for the store day array used in process_store_day_info.

write_store_day_data()

This function will write all elements of the store day array to the store day output file. The file format for the store day file can be found in the I/O section of this document. The information should be ordered by store (alphabetically not numerically) and business date.

process_codes_info()

This function will query codes information from the code_detail table and uses this information to populate the codes array. This function also calls size_codes_arrays() to allocate memory for the codes array. The columns selected in this process include code, code_type, and code_seq. The information should be ordered by code_type and code. All records in the codes array should be written to the codes output file by calling write_codes_data().

size_codes_arrays()

This function allocates memory for the codes array used in process_codes_info.

write_codes_data()

This function will write all elements of the codes array to the codes output file. The file format for the codes file can be found in the I/O section of this document. This information should be ordered by code_type and code.

process_error_info()

This function will query error code information from the sa_error_codes table and uses this information to populate the errors array. This function also calls size_error_arrays() to allocate memory for the error array. The columns selected in this process include error_code, error_desc, and rec_solution (recommended solution). The information should be ordered by error_code. All records in the errors array should be written to the errors output file by calling write_error_data().

size_error_arrays()

This function allocates memory for the error array used in process_error_info.

write_error_data()

This function will write all elements of the error array to the error output file. The file format for the error file can be found in the I/O section of this document. This information should be ordered by error_code.

process_cc_val_info()

This function will query credit card validation information from the sa_cc_val table and uses this information to populate the credit card validation array. This function also calls size_cc_val_arrays() to allocate memory for the credit card validation array. The columns selected in this process include length, from_prefix, to_prefix, tender_type_id, and value type. The information should be ordered by length (alphabetically not numerically) and from_prefix. All records in the credit card validation array should be written to the credit card validation output file by calling write_cc_val_data().

size_cc_val_arrays()

This function allocates memory for the credit card valdiation array used in process_cc_val_info.

write_cc_val_data()

This function will write all elements of the credit card validation array to the credit card validation output file. The file format for the credit card validation file can be found in the I/O section of this document. This information should be ordered by length (alphabetically not numerically) and from_prefix.

process_store_pos_info()

This function will query store POS information from the sa_store_pos table and uses this information to populate the store POS array. This function also calls size_store_pos_arrays() to allocate memory for the store POS array. The columns selected in this process include store, pos_type, start_tran_no, and end_tran_no. The information should be ordered by store (alphabetically not numerically) and pos_type. All records in the store POS array should be written to the store POS output file by calling write_store_pos_data().

size_store_pos_arrays()

This function allocates memory for the store POS array used in process_store_pos_info.

write_store_pos_data()

This function will write all elements of the store POS array to the store POS output file. The file format for the store POS file can be found in the I/O section of this document. This information should be ordered by store (alphabetically not numerically) and pos_type.

process_tender_type_info()

This function will query tender type information from the pos_tender_type_head table and uses this information to populate the tender type array. This function also calls size_tender_type_arrays() to allocate memory for the tender type array. The columns selected in this process include tender_type_group, tender_type_id, and tender_type_desc. The information should be ordered by tender_type_group and tender_type_id (alphabetically not numerically). All records in the tender array should be written to the tender type output file by calling write_tender_type_data().

size_tender_type_arrays()

This function allocates memory for the tender type array used in process_tender type_info.

write_tender_type_data()

This function will write all elements of the tender type array to the tender type output file. The file format for the tender type file can be found in the I/O section of this document. This information should be ordered by tender_type_group and tender_type_id (alphabetically not numerically).

process_merch_codes_info()

This function will query merch code information from the non_merch_code_head table and uses this information to populate the merch codes array. This function also calls size_merch_codes_arrays() to allocate memory for the merch codes array. The columns selected in this process include non_merch_code. The information should be ordered by non_merch_code. All records in the merch codes array should be written to the merchant codes output file by calling write_merch_codes_data().

size_merch_codes_arrays()

This function allocates memory for the merch codes array used in process_merch_codes_info.

write_merch_codes_data()

This function will write all elements of the merch codes array to the merchant codes output file. The file format for the merchant codes file can be found in the I/O section of this document. This information should be ordered by non_merch_code.

process_partner_info()

This function will query partner information from the partner table and uses this information to populate the partner array. This function also calls size_partner_arrays() to allocate memory for the partner array. The columns selected in this process include partner_type, and partner_id. The information should be ordered by partner_id. All records in the partner array should be written to the partner output file by calling write_partner_data().

size_partner_arrays()

This function allocates memory for the partner array used in process_partner_info.

write_partner_data()

This function will write all elements of the partner array to the partner output file. The file format for the partner file can be found in the I/O section of this document. This information should be ordered by partner_id.

process_supplier_info()

This function will query supplier information from the sups table and uses this information to populate the supplier array. This function also calls size_supplier_arrays() to allocate memory for the supplier array. The columns selected in this process include supplier, and sups_status. The information should be ordered by supplier (alphabetically not numerically). All records in the supplier array should be written to the supplier output file by calling write_supplier_data().

size_supplier_arrays()

This function allocates memory for the supplier array used in process_supplier_info.

write_supplier_data()

This function will write all elements of the supplier array to the supplier output file. The file format for the supplier file can be found in the I/O section of this document. This information should be ordered by supplier (alphabetically not numerically).

process_employee_info()

This function will query employee information from the sa_store_emp table and uses this information to populate the employee array. This includes all stores where pos_id is NOT NULL. This function also calls size_employee_arrays() to allocate memory for the employee array. The columns selected in this process include store, pos_id, and emp_id. The information should be ordered by store (alphabetically not numerically) and pos_id. All records in the employee array should be written to the employee output file by calling write_employee_data().

size_employee_arrays()

This function allocates memory for the employee array used in process_employee_info.

write_employee_data()

This function will write all elements of the employee array to the employee output file. The file format for the employee file can be found in the I/O section of this document. This information should be ordered by store (alphabetically not numerically) and pos_id.

process_banner_info()

This function will query banner information from the store, banner and channels tables and uses this information to populate the banner array. This function also calls size_banner_arrays() to allocate memory for the banner array. The columns selected in this process include store, and banner_id. The information should be ordered by store (alphabetically not numerically). All records in the banner array will be written to the banner output file by calling write_banner_data().

size_banner_arrays()

This function allocates memory for the banner array used in process_banner_info.

write_banner_data()

This function will write all elements of the banner array to the banner output file. The file format for the banner file can be found in the I/O section of this document. This information should be ordered by store (alphabetically not numerically).

final()

This function will terminate restart-recovery. It also calls retek_refresh_thread() to refresh the current thread.

Output Specifications

As all files produced by this program are used only internally, they consist of only detail records.

Char field types are left justified and blank padded.

Number field types are right justified and zero padded.

Record Name	Field Name	Field Type	Default Value	Description
Item Data	Item	char(25)		Unique item identifier
	Dept	char(4)		Department identifier
	Class	char(4)		Class identifier
	Subclass	char(4)		Subclass identifier
	Standard_uom	char(4)		Standard UOM
	Catch_weight_ind	char(1)		Catch weight indicator
Waste Data	Item	char(25)		Unique item identifier
	Waste_type	char(6)		Waste type identifier
	Waste_pct	number(16)		Waste percent
Ref Item Data	Item_parent	char(25)		Item Parent
	Item	char(25)		Unique item identifier
Primary Variant Data	Item_loc	number(10)		Item location
	Item	char(25)		Unique item identifier
	Primary_variant	char(25)		Primary variant
Variable UPC Data	Format_id	char(1)		Format identifier
	Format_desc	char(20)		Format description
	Prefix_length	number(1)		Prefix length
	Begin_item_digit	number(2)		Determines the first digit of the item number.
	Begin_var_digit	number(2)		Determines the first digit of the variable weight/price.
	Check_digit	number(2)		Position of the check digit.
	Prefix	number(2)		Item master prefix
Store Day Data	Store	number(10)		Store number

Record Name	Field Name	Field Type	Default Value	Description
	Business_date	char(8)		Business date – format: YYYYMMDD
	Store_day_seq_no	number(20)		Unique store/day identifier
	Day	number(3)		Day
	Tran_no_generated	char(6)		If NULL then blank
	System_code	char(6)		System code
	Currency_rtl_dec	number(1)		Currency retail decimal places
Code Data	Code_type	char(4)		Unique code type identifier
	Code	char(6)		Unique code identifier
	Code_seq	number(4)		Unique code sequence identifier
Error Data	Error_code	char(25)		Error identifier
	Error_desc	char(255)		Error description
	Rec_solution	char(255)		Recommended solution (If NULL then ‘there is no solution’)
Credit Card Validation Data	Length	number(2)		Card number length
	From_prefix	number(6)		Start value for range of valid prefixes.
	To_prefix	number(6)		End value for range of valid prefixes.
	Tender_type_id	number(6)		Credit card ID
	Val_type	char(6)		Validation type. If NULL, than use “NONE”.
Store POS Data	Store	number(10)		Store identifier
	Pos_type	char(6)		POS type identifier
	Start_tran_no	number(10)		First transaction number produced. Right justified and zero padded.
	End_tran_no	number(10)		Last transaction number produced. Right justified and zero padded.
Tender Type Data	Tender_type_group	char(6)		Tender type group
	Tender_type_id	number(6)		Tender type identifier. Right justified and zero padded.

Record Name	Field Name	Field Type	Default Value	Description
	Tender_type_desc	char(40)		Tender type description.
Merchant Code Data	Non_merch_code	char(6)		Code identifying a non-merchandise cost that can be added to an invoice.
Partner Data	Partner_type	char(6)		Specifies the type of partner. Valid values are Bank 'BK', Agent 'AG', Freight Forwarder 'FF', Importer 'IM', Broker 'BR', Factory 'FA', Applicant 'AP', Consolidator 'CO', and Consignee 'CN', Supplier hierarchy level 1 'S1', Supplier hierarchy level 2 'S2', Supplier hierarchy level 3 'S3'.
	Partner_id	char(10)		Partner vendor number.
Supplier Data	Supplier	number(10)		Supplier vendor number.
	Sup_status	char(1)		Determines whether the supplier is currently active. Valid values include: 'A' for an active supplier or 'I' for an inactive supplier.
Employee Data	Store	number(10)		Store number.
	Pos_id	char(10)		The POS ID of the employee.
	Emp_id	char(10)		The employee ID of the employee.
Banner Data	Store	number(10)		Store number
	Banner_id	number(4)		Banner identifier

Scheduling Considerations

Processing Cycle:	Anytime – Sales Audit is a 24/7 system.
Scheduling Diagram:	This module should be executed in the earliest phase, before the first import of RTLOGs into ReSA.
Pre-Processing:	sastdycr.pc
Post-Processing:	saimptlog.pc
Threading Scheme:	N/A

Restart Recovery

Restart recovery does not apply in the typical sense because sagetref writes to output files and will not need to have restart capabilities, however restart is used for bookmarking purposes.

Vendor Invoicing for Complex Deals [vendinvc]

Design Overview

This batch module creates records in staging tables dealing with complex type deals.

The invoicing logic will be driven from the billing period estimated next invoice date for complex deals. The amount to be invoiced will be the sum of the income accruals of the deal since the previous invoice date (or the deal start date for the first collection).

The processing will be as follows:

1 Write a header record to the holding table for the deal

2 Determine which reporting periods are to be invoiced

3 For Complex Deals Aggregate the income for the reporting periods

4 Write a deal detail record to the holding table for each item, location

5 Update the next invoice date to vdate, and update estimated next date

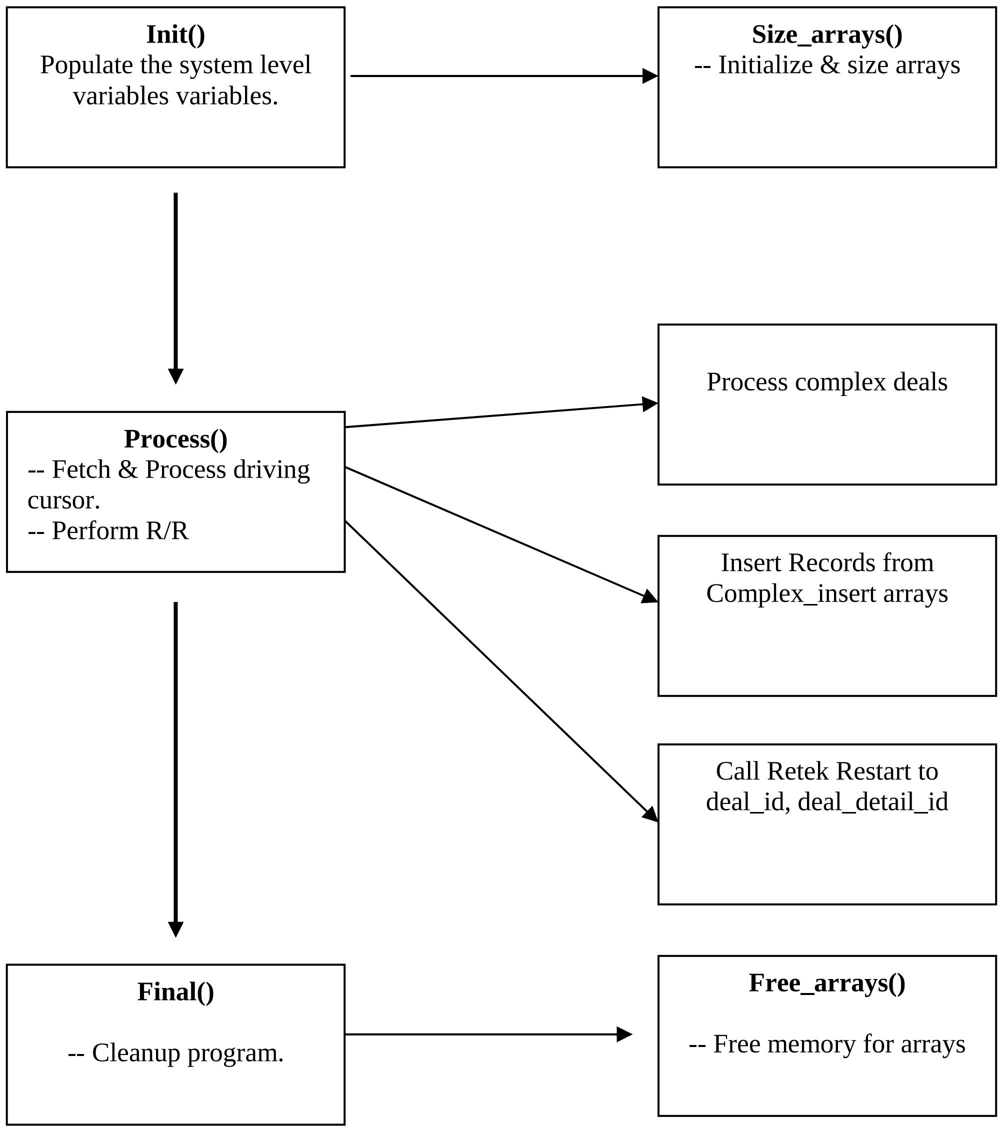
Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No
VENDINVC_TEMP	No	Yes	No	No	No
DEAL_HEAD	No	Yes	No	Yes	No
DEAL_ACTUALS_ITEM_LOC	No	Yes	No	No	No
STORE	No	Yes	No	No	No
WH	No	Yes	No	No	No
VAT_ITEM	No	Yes	No	No	No
DEAL_ACTUALS_FORECAST	No	Yes	No	No	No
STAGE_COMPLEX_DEAL_HEAD	No	No	Yes	No	No
STAGE_COMPLEX_DEAL_DETAIL	No	No	Yes	No	No

Stored Procedures / Shared Modules (Maintainability)

DEAL_FINANCE_SQL.CALC_NEXT_REPORT_DATE - Function to get the next reporting date

Program Flow


Init()
Populate the system level variables variables.
Size_arrays()
-- Initialize & size arrays
Process complex deals
Process()
-- Fetch & Process driving cursor.
-- Perform R/R
Insert Records from Complex_insert arrays
Call Retek Restart to deal_id, deal_detail_id
Final()
-- Cleanup program.
Free_arrays()
-- Free memory for arrays

Function Level Description

main()

Validate the program arguments and logon to Oracle.

Call the init() function to initialize restart / recovery and variables.

Call the process() function to execute main program logic.

Call the final() function to clean up all internal processing.

init()

Call standard retek initialization function retek_init() to initialize restart / recovery.

Gets the following system level variables (program variables):

- SYSTEM_OPTIONS.CURRENCY_CODE SYSTEM_VARIABLES.LAST_EOM_DATE
- SYSTEM_VARIABLES.NEXT_EOM_DATE – 7
- PERIOD.VDATE (ps_vdate)
- SYSTEM_OPTIONS.VAT_IND

final()

Free all arrays by calling function free_arrays().

Call standard retek close function retek_close().

process()

Initialize the array structures by calling size_arrays() .

Call out get_location_info().

In a while loop fetch required information from the driving cursor c_get_deals and fetch into the array structure pa_complex_fetch.

If this is the first found invoicable reporting period for this location, item, order no, call out check_date().

Process complex deals by calling out the following functions: calculate_start_invoice_date(),post_complex_head(),insert_complex_head(),insert_complex_detail(),update_deal_head() and post_complex_detai().

Call retek_force_commit() to commit the entries.

Driving Cursor

```
      EXEC SQL DECLARE c_get_deals CURSOR FOR
         SELECT /*+ no_expand */ vt.deal_id,
                vt.deal_detail_id,
                vt.item,
                vt.location,
                vt.loc_type,
                vt.order_no,
                NVL(vt.actual_income, 0),
                NVL(vt.actual_turnover_units, 0),
                NVL(vt.actual_turnover_revenue, 0),
                TO_CHAR(vt.reporting_date, 'YYYYMMDDHH24MISS'),
                vt.bill_back_period,
                vt.deal_reporting_level,
                vt.active_date,
                vt.close_date,
                DECODE(vt.partner_type, 'S', vt.supplier,
   vt.partner_id),
                vt.partner_type,
                vt.currency_code,
                vt.bill_back_method,
                vt.invoice_processing_logic,
                vt.include_vat_ind
           FROM vendinvc_temp vt
          WHERE MOD(vt.deal_id, TO_NUMBER(:ps_restart_num_threads)) + 1
   = TO_NUMBER(:ps_restart_thread_val)
            AND (   vt.deal_id > nvl(:ps_restart_deal_id, -999)
                 OR
                   (   vt.deal_id        = NVL(:ps_restart_deal_id, -
   999)
                   AND vt.deal_detail_id >
   NVL(:ps_restart_deal_detail_id, -999))
                )
          ORDER BY vt.deal_id ASC,
                   vt.deal_detail_id ASC,
                   vt.location ASC,
                   vt.item ASC,
                   vt.order_no,
                   vt.reporting_date DESC;
```

check_date()

All reporting dates that are to be summed for a given invoicing period must fit with the following rules, however as the calling cursor is ordered by reporting date DESC this routine need only be called for the first reporting date for each location.

For a Weekly Reporting Period against a Weekly Invoicing Period you can not invoice the last week in the month until the EOM has closed, so check if the reporting date is within 8 days of the Next EOM date.

For a Monthly Reporting Period against a Monthly, Quarterly, Half Yearly or Annual Invoicing Period you can not invoice until after EOM, so check if the reporting is on or before EOM.

calculate_start_invoice_date()

Calculate the START_INVOICE_DATE as being the Day after the Reporting Date prior to the earliest Reporting Date selected for invoicing OR if the earliest Reporting Date selected is the first for the deal, then the deal start date.

get_location_info()

Cursor c_get_location_info retrieves all location data from STORE and WH tables.

Loop on location_info cursor, copy the values to the holding array pa_fetch_loc_info.

Copy fetched data from pa_fetch_loc_info into array structure pa_loc_info.

if pa_fetch_loc_info.i_vat_region_ind = 0, then pa_loc_info.i_vat_region_ind = -1.

Validate the array size and complete a resize as required.

get_loc_details()

Check if the current location is the same as the previous location, if so set the vat region to be the same. If the location has changed, find the new vat region, new location and loc type.

post_complex_head()

Populate the current record in the insert array pa_complex_head_insert, data comes from the values passed to the function.

Call the stored procedure DEAL_FINANCE_SQL.CALC_NEXT_REPORT_DATE() and populate the complex insert (and update) array with the return value

Validate the array size and complete a resize as required

post_complex_detail()

Populate the current record in the insert array pa_complex_detail_insert, data comes from the values passed to the function.

Get the location details by calling function get_loc_details()

If the local currency and the deal currency are different convert the amount by calling library function convert().

Validate the array size and complete a resize as required.

insert_complex_head ()

Insert the contents of the holding array pa_complex_head_insert into the fixed deal staging table, STAGE_COMPLEX_DEAL_HEAD.

insert_complex_detail ()

Insert the contents of the holding array pa_complex_detail_insert into the fixed deal staging table, STAGE_COMPLEX_DEAL_DETAIL.

update_deal_head()

Update the last_invoice_date, last_update_datetime and est_next_invoice date on deal head for the invoiced deals from the complex head update structure, pa_complex_head_insert.

size_arrays ()

Allocate memory for elements of following structures : - driving cursor fetch array - pa_complex_fetch, pa_complex_head_insert, pa_complex_detail_insert, pa_fetch_loc_info and pa_loc_info.

resize_arrays ()

Use the memory allocation macro to allocate memory for the elements of following structures:- driving cursor fetch array - pa_complex_head_insert, pa_complex_detail_insert and pa_loc_info.

free_arrays ()

Uses the memory deallocation macro to free the memory used by the elements of the following structures:- driving cursor fetch array - pa_complex_fetch, pa_complex_head_insert, pa_complex_detail_insert and pa_loc_info..

Scheduling Considerations

Processing Cycle:	Ad-Hoc. Must be run before salmnth, after dealact and before the new programs which perform forecast processing and DAILY_DATA roll up.
Scheduling Diagram:	N/A - The program should be run daily
Pre-Processing:	Truncate STAGE_FIXED_DEAL_HEAD and STAGE_FIXED_DEAL_DETAIL tables. (vendinvf_pre)
Post-Processing:	Call out process_deal_head() function to update status of the fixed_deal to ‘I’. (vendinvf_post)
Threading Scheme:	N/A

Restart Recovery

The Logical Unit of Work (LUW) for the program is a transaction consisting of deal_id, deal_detail_id.

Vendor Invoicing for Fixed Deals [vendinvf]

Design Overview

This batch module creates records in staging tables dealing with fixed type deals.

The invoicing logic will be driven by the collection dates for fixed deals. The amount to be invoiced will be retrieved directly from fixed deal tables for a given deal date.

The processing will be as follows:

1 Write a header record to the holding table for the deal

2 Determine which reporting periods are to be invoiced

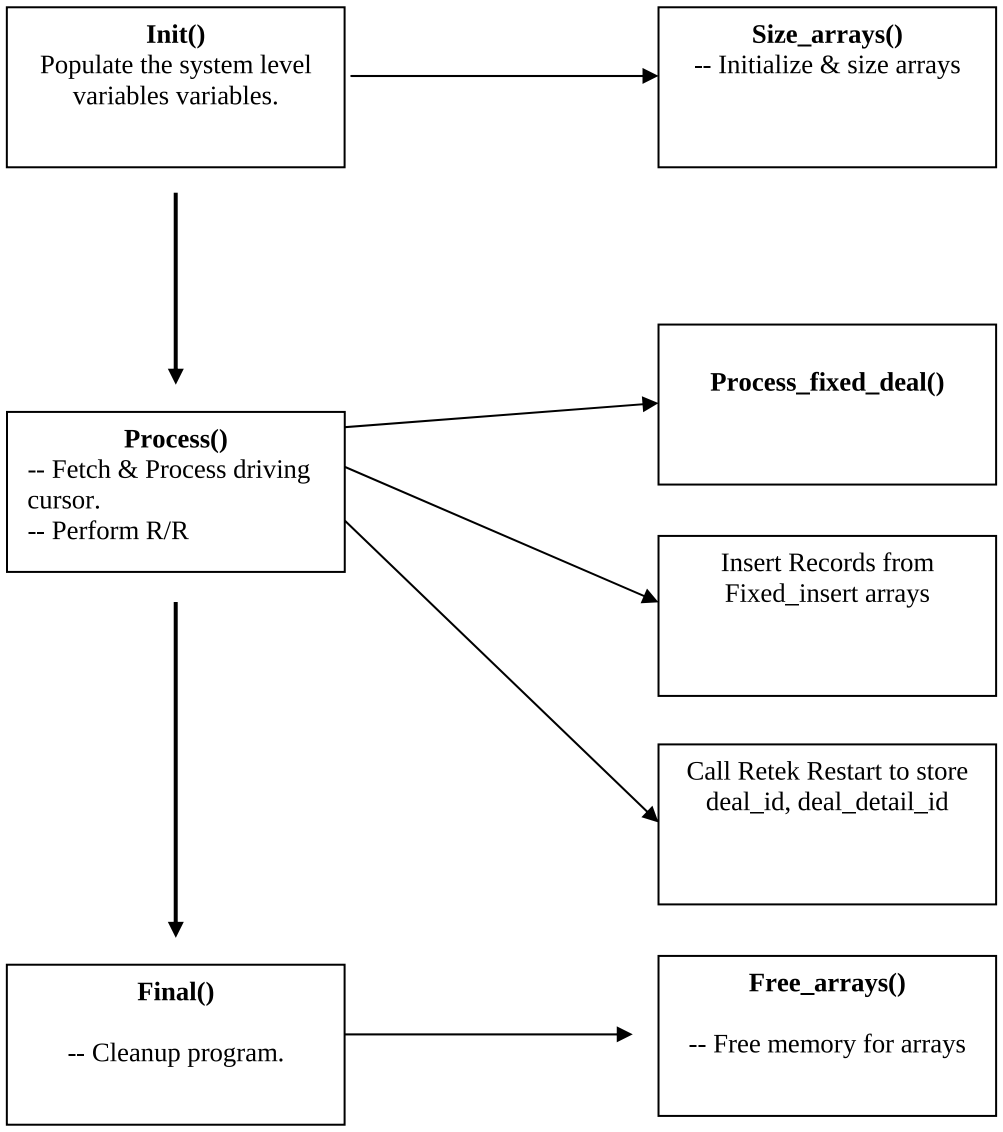
3 Distribute the income for the reporting period across location / subclass

4 Write a deal detail record to the holding table for each location / subclass

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
FIXED_DEAL	No	Yes	No	No	No
FIXED_DEAL_DATES	No	Yes	No	No	No
FIXED_DEAL_MERCH	No	Yes	No	No	No
FIXED_DEAL_MERCH_LOC	No	Yes	No	No	No
SUBLASS	No	Yes	No	No	No
STAGE_FIXED_DEAL_HEAD	No	No	Yes	No	No
STAGE_FIXED_DEAL_DETAIL	No	No	Yes	No	No

Program Flow


Init()
Populate the system level variables variables.
Size_arrays()
-- Initialize & size arrays
Process_fixed_deal()
Process()
-- Fetch & Process driving cursor.
-- Perform R/R
Insert Records from Fixed_insert arrays
Call Retek Restart to store deal_id, deal_detail_id
Final()
-- Cleanup program.
Free_arrays()
-- Free memory for arrays

Function Level Description

main()

Validate the program arguments and logon to Oracle.

Call the init() function to initialize restart / recovery and variables

Call the process() function to execute main program logic

Call the final() function to clean up all internal processing

init()

Call standard retek initialization function retek_init() to initialize restart / recovery.

Gets the following system level variables (program variables):

- SYSTEM_OPTIONS.CURRENCY_CODE
- SYSTEM_OPTIONS.VAT_IND
- PERIOD.VDATE (ps_vdate)

final()

Free all arrays by calling function free_arrays().

Call standard retek close function retek_close().

process()

Initialize the array structures by calling size_arrays().

In a while loop fetch required information from the driving cursor c_get_deals and fetch into the array structure pa_deals_fetch.

Call out process_fixed_deal().

Insert the data that is buffered in the Insert Arrays and reset the Insert record counts by insert_fixed_head() and insert_fixed_detail().

Call retek_force_commit() to commit the entries.

Driving Cursor

```
    EXEC SQL DECLARE c_get_deals CURSOR FOR

       SELECT fd.deal_no
deal_id,

              TO_CHAR(fdd.collect_date, 'YYYYMMDDHH24MISS')
inv_period_start,

              fd.merch_ind
merch_ind,

              fdd.fixed_deal_amt
fd_amount,

              DECODE(fd.partner_type, 'S', fd.supplier,
fd.partner_id)  vendor,

              fd.partner_type
vendor_type,
                /* Retek CR required to get the currency code from */
                /*  fixed_deal once it has been added to the table */
                :ps_currency
currency_code,
                fd.deb_cred_ind
deb_cred_ind,
                fd.invoice_processing_logic
invoice_processing_logic,
                fd.non_merch_code
non_merch_code,
                fd.vat_ind
vat_ind,
                fd.vat_code
vat_code,
                fd.vat_rate
vat_rate
           FROM fixed_deal fd,
                fixed_deal_dates fdd
          WHERE fd.status    = 'A'
            AND fd.deal_no   = fdd.deal_no
            AND TO_DATE(:ps_vdate, 'YYYYMMDDHH24MISS') =
fdd.collect_date
            AND fd.invoice_processing_logic != 'NO'
            AND MOD(fd.deal_no, TO_NUMBER(:ps_restart_num_threads)) + 1
= TO_NUMBER(:ps_restart_thread_val)
         ORDER BY fd.deal_no;
```

process_fixed_deal()

Contains a cursor c_get_fixed and identifies a set of potential Fixed Deals for invoicing.

Process the Merchandise deal types from the global driving cursor fetched array ps_deal_fetch.

Loop through the fetched cursor pa_fixed_fetch.

Call out post_fixed_head to post the fixed deal header record.

Validate if the current deal merchandise Indicator = ‘Y’, if ‘Y’ then call the function local_calculate_income() then post the calculated income record to the fixed detail insert array, pa_fixed_fetch by calling function post_fixed_detail().

If the current deal merchandise Indicator = ‘N’, No details need for non-merchandised deals, so there’s no point in proessing the remaining records

Exit the Loop of the fetched cursor pa_fixed_fetch.

Process non-Merchandise deal type from the global driving cursor fetched array ps_deal_fetch.

Loop through the fetched cursor pa_fixed_fetch.

Call the function post_fixed_head().

Because of Processing for a merchandised Fixed Deal, the final total needs to be corrected by calling function correct_fixed_total().

post_fixed_head()

Populate the next record in the insert array pr_fixed_head_insert, data comes from the current fixed deal cursor record, pa_fixed_fetch.

With nullable merch_ind and want non_merch_ind, validate null and reverse "Y" and "N".

Check the system vat and local vat indicators are 'Y', if yes set to the current fixed deal cursor record, pa_fixed_fetch, else set to -1.

post_fixed_detail()

Populate the next record in the insert array pr_fixed_detail_insert, data comes from the current fixed deal cursor record, pa_fixed_fetch.

The insert array may need to be resized if full.

correct_fixed_total()

To allow for rounding errors, etc. and ensure that the total amount distributed across the location hierarchy is equal to the total amount available for the fixed deal period, a correction is applied to the last location within a group. So, this becomes the fixed deal total minus the sum of the amounts allocated to the other locations.

Adjust the income amount the most recently posted fixed detail record new fixed_deal.total_amount = fixed_deal.total_amount + (fixed_head.total_amount - total_to_date).

insert_fixed_head ()

Insert the contents of the holding array pa_fixed_head_insert into the fixed deal staging table, STAGE_FIXED_DEAL_HEAD.

insert_fixed_detail ()

Insert the contents of the holding array pa_fixed_detail_insert into the fixed deal staging table, STAGE_FIXED_DEAL_DETAIL.

local_calculate_income()

Populate the fixed deal calculate income structure from the current record of the fixed deal array.

Fixed deal calculate income structure passed to the calculate_income function calculate_income().

Update the current fixed deal array record with the income amount from calculate income.

calculate_income()

Note: This is the same function as used in dealfinc.pc This function will calculate the income for each deal/dept/class/subclass/ location split.

First count if the deal or the subclass has changed, or first time by fetching cursor c_subclass_count.

Calculate the factor for the whole deal, this is used for pro-rating at location level by fetching cursor c_multipler_factor. Only do this for a change of deal, or first time.

Calculate the amount for this dept / class / subclass / location.

Keep the cumulative total for this deal.

Keep track of the previous deal_no and seq_no by copying to variables.

size_arrays ()

Allocate memory for elements of following structures : - driving cursor fetch array - pa_deals_fetch, pa_fixed_fetch, pa_fixed_head_insert and pa_fixed_detail_insert.

resize_arrays ()

Use the memory allocation macro to allocate memory for the elements of following structures:- driving cursor fetch array - pa_fixed_head_insert and pa_fixed_detail_insert.

free_arrays ()

Uses the memory deallocation macro to free the memory used by the elements of the following structures:- driving cursor fetch array - pa_deals_fetch, pa_fixed_fetch, pa_fixed_head_insert and pa_fixed_detail_insert.

Scheduling Considerations

Processing Cycle:	Ad-Hoc. Must be run before salmnth, after dealact and before the new programs which perform forecast processing and DAILY_DATA roll up.
Scheduling Diagram:	N/A - The program should be run daily
Pre-Processing:	Truncate STAGE_COMPLEX_DEAL_DETAIL, STAGE_COMPLEX_DEAL_HEAD and VENDINVC_TEMP tables. Then insert into VENDINVC_TEMP. (vendinvc_pre)
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

The Logical Unit of Work (LUW) for the program is a transaction consisting of deal_id, deal_detail_id.