Oracle® Retail Merchandising System

Operations Guide Addendum
Release 10.2.7

December 2008

ORACLE

Oracle® Retail Merchandising System Operations Guide Addendum, Release 10.2.7

Primary Author: Nathan Young

Value-Added Reseller (VAR) Language

Oracle Retail VAR Applications

The following restrictions and provisions only apply to the programs referred to in this section and licensed to you. You acknowledge that the programs may contain third party software (VAR applications) licensed to Oracle. Depending upon your product and its version number, the VAR applications may include:

(i) the software component known as **ACUMATE** developed and licensed by Lucent Technologies Inc. of Murray Hill, New Jersey, to Oracle and imbedded in the Oracle Retail Predictive Application Server – Enterprise Engine, Oracle Retail Category Management, Oracle Retail Item Planning, Oracle Retail Merchandise Financial Planning, Oracle Retail Advanced Inventory Planning and Oracle Retail Demand Forecasting applications.

(ii) the **MicroStrategy** Components developed and licensed by MicroStrategy Services Corporation (MicroStrategy) of McLean, Virginia to Oracle and imbedded in the MicroStrategy for Oracle Retail Data Warehouse and MicroStrategy for Oracle Retail Planning & Optimization applications.

(iii) the **SeeBeyond** component developed and licensed by Sun MicroSystems, Inc. (Sun) of Santa Clara, California, to Oracle and imbedded in the Oracle Retail Integration Bus application.

(iv) the **Wavelink** component developed and licensed by Wavelink Corporation (Wavelink) of Kirkland, Washington, to Oracle and imbedded in Oracle Retail Store Inventory Management.

(v) the software component known as **Crystal Enterprise Professional and/or Crystal Reports Professional** licensed by Business Objects Software Limited ("Business Objects") and imbedded in Oracle Retail Store Inventory Management.

(vi) the software component known as **Access Via™** licensed by Access Via of Seattle, Washington, and imbedded in Oracle Retail Signs and Oracle Retail Labels and Tags.

(vii) the software component known as **Adobe Flex™** licensed by Adobe Systems Incorporated of San Jose, California, and imbedded in Oracle Retail Promotion Planning & Optimization application.

(viii) the software component known as **Style Report™** developed and licensed by InetSoft Technology Corp. of Piscataway, New Jersey, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

(ix) the software component known as **DataBeacon™** developed and licensed by Cognos Incorporated of Ottawa, Ontario, Canada, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

Contents

Preface

Oracle Retail Operations Guides are designed so that you can view and understand the application's 'behind-the-scenes' processing, including such information as the following:

- Key system administration configuration settings
- Technical architecture
- Functional integration dataflow across the enterprise

This Operations Guide Addendum should be used in conjunction with previously released Oracle Retail Merchandising System 10.x documentation.

Audience

Anyone with an interest in developing a deeper understanding of the underlying processes and architecture supporting Oracle Retail Merchandising System functionality will find valuable information in this guide. There are three audiences in general for whom this guide is written:

- Business analysts looking for information about processes and interfaces to validate the support for business scenarios within and other systems across the enterprise.
- System analysts and system operations personnel:
  - Who are looking for information about Oracle Retail Merchandising System's processes internally or in relation to the systems across the enterprise.
  - Who operate Oracle Retail Merchandising System regularly.
- Integrators and implementation staff with overall responsibility for implementing Oracle Retail Merchandising System.

Related Documents

For more information, see the following documents in the Oracle Retail Merchandising System Release 10.2.7 documentation set:

- *Oracle Retail Merchandising System Release Notes*
- *Oracle Retail Merchandising System Installation Guide*

Customer Support

https://metalink.oracle.com

When contacting Customer Support, please provide the following:

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

Review Patch Documentation

If you are installing the application for the first time, you install either a base release (for example, 13.0) or a later patch release (for example, 13.0.2). If you are installing a software version other than the base release, be sure to read the documentation for each patch release (since the base release) before you begin installation. Patch documentation can contain critical information related to the base release and code changes that have been made since the base release.

Oracle Retail Documentation on the Oracle Technology Network

In addition to being packaged with each product release (on the base or patch level), all Oracle Retail documentation is available on the following Web site (with the exception of the Data Model which is only available with the release packaged code):

http://www.oracle.com/technology/documentation/oracle_retail.html

Documentation should be available on this Web site within a month after a product release. Note that documentation is always available with the packaged code on the release date.

Conventions

Navigate: This is a navigate statement. It tells you how to get to the start of the procedure and ends with a screen shot of the starting point and the statement "the Window Name window opens."

> **Note:** This is a note. It is used to call out information that is important, but not necessarily part of the procedure.

```
This is a code sample
    It is used to display examples of code
```

A hyperlink appears like this.

1
Introduction

The information in this document reflects modifications and updates to the *Oracle Retail Merchandising System 10.0 Operations Guide* and any subsequent RMS 10.x.x Operations Guide Addendums. Using this document in conjunction with the *Oracle Retail Merchandising System 10.0 Operations Guide* provides retailers with a complete overview of the application.

The following batch designs have been updated for the RMS 10.2.7 release:

- vrplbld (Vendor Replenished Order Build)
- ediupack (EDI Supplier Order Acknowledgements and Changes)

For more specific information regarding enhancements and modifications made to the previous Oracle Retail Merchandising System release, see the *Oracle Retail Merchandising System 10.2.7 Release Notes*.

2

Batch Designs

Retailers should refer to these sections in lieu of the corresponding batch designs in the RMS 10.0 Operations Guide or any subsequent RMS 10.x.x Operation Guide Addendums.

Batch designs describe how, on a technical level, an individual batch module works and the database tables that it affects. In addition, batch designs contain file layout information that is associated with the batch process.

ediupack (EDI Supplier Order Acknowledgements and Changes)

Design Overview

This program has four functions: to acknowledge vendor receipt of a buyer generated order without changes, to acknowledge vendor receipt of a buyer generated order with date, cost or quantity modifications, to notify buyer of a vendor-generated order, and to acknowledge order cancellations.

All acknowledgements update the ordhead table with acknowledgement information.

When the supplier sends the acknowledgement with modifications, they can send the entire purchase order or only the changes. The file details are matched to the current order. If the Not Before Date, Not After Date, Quantity, Price, and item all match the current order, then no changes were submitted. If one of the variables is blank, for example the price, assume that no pricing changes were made. As soon as one of the variables does not match, the order has been changed. These changes are not written directly to the order; they are written to the revision tables. Revisions are accepted in the ordering dialog and changed orders will be resubmitted via edidlord. The revision number is the incremented value of the last (or maximum) revision number for revisions of the order on the revision tables.

Vendor generated orders create new orders by inserting new records on the EDI temporary order tables.

This program uses linked list processing. The linked list should hold one transaction at a time (THEAD, transaction header, through TTAIL, transaction tail). If there is an error with a transaction set, that transaction is written to the reject file so it can be fixed and reprocessed later, then the program continues to the next transaction set.

Vendor generated orders:

Item, location, supplier, and cost information are all required for this transaction type. Oracle Retail only supports single shipping locations for vendor replenishment. Once all required segments are read and validated, order information is written to the edi_ord_temp table. In addition, in a multichannel environment the supplier's channel ID must correspond to a virtual warehouse in the physical being shipped to.

Acknowledgement without change:

Items that are read are read and validated against the current order information. The ordhead table is updated to reflect that the PO acknowledgement was received.

Acknowledgment with change:

The following fields can be modified on an existing buyer generated order: the not before date, not after date, item cost, qty ordered, and order location (for multiple shipping locations only). Date information that is sent is compared to the dates stored on the ordhead table. If a difference is detected, a revision is written to the ordhead_rev table for the order. Cost information that is sent is compared to the cost stored on the ordloc tables. If a difference is detected then revisions are written for the item and all of its shipping locations with the new cost information. Quantity information is compared with the information on the ordloc table. If a difference is detected in the sent quantity and the stored quantities, then a record is inserted/updated on ordloc_rev with the new order quantity.

New items and shipping locations can also be sent. New item information is added to the ordsku_rev and the ordloc_rev tables. New locations for existing items are inserted to the ordloc_rev tables.

If the Batch with Online Users Indicator is 'Y', it checks if the records to be updated are locked. Information of the locked records is inserted to batch_lock_log table.

For cross-dock VMI PO creation, it allows having multiple shipping locations for same item/supplier combination. Among multiple shipping locations, one is source warehouse and the other can be store or warehouse, which is sourced from source warehouse. The quantity given for source warehouse in TSHIP record should be greater than or equal to sum of quantities given in TSHIP record for cross docked locations.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
EDI_ORD_TEMP	No	Yes	Yes	Yes	No
ITEM_MASTER	Yes	Yes	No	No	No
ITEM_SUPPLIER	Yes	Yes	No	No	No
ORDHEAD_REV	Yes	No	Yes	Yes	No
ORDSKU_REV	Yes	No	Yes	Yes	No
ORDLOC_REV	Yes	No	Yes	Yes	No
BATCH_LOCK_LOG	No	Yes	Yes	No	Yes

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc in Phases 1,2, or 3
Scheduling Diagram	N/A
Pre-Processing	N/A
Post-Processing	Before Open to Buy processing
Threading Scheme	N/A

Restart Recovery

The files will not have enough volume to warrant the implementation of restart recovery for commit/rollback considerations but minimal restart/recovery capability will be added.

Program Flow

N/A

Shared Modules

- NEW_ITEM_LOC
- NEW_PACK_LOC
- PRICING_ATTRIB_SQL.BUILD_PACK_RETAIL
- SUPP_ITEM_ATTRIB_SQL.GET_ATTRIB_FOR_ORDSKU

Function Level Description

Init:

Get vdate from period; open input file and reject file.

It retrieves the btch_w_usr_ind from the system_options table to determine users can be online at any time. If the Batch with Online Users indicator is set to 'Y', ediupack.pc's records in the batch_lock_log are deleted if the batch is run for the first time.

Process

This function reads and validates the transaction header. Then the item information lines and shipment location lines are read and processed, if present.

The ordhead table is updated to reflect that a PO acknowledgement has been received after the transaction has been processed.

get_line:

Read a line from the input file and add it to the linked list.

dump_to_reject:

Read in the rest of a rejected transaction and write it to the reject file.

validate_THEAD

This function processes the THEAD line for purchase order acknowledgement. Variables on the line are validated.

If the acknowledgement is a vendor generated order (vendor replenishment) set acknowledgement type and changed order flags. No more validation or processing occurs in this function for vendor generated orders.

Buyer order acknowledgement, with or without changes, retrieve the supplier number and the acknowledgement flag from ordhead. The edi acknowledgement must not have been received, and the supplier is validated with the supplier value supplied in the file (if present). If the order is being changed (type AC) then the next revision number is retrieved from the ordloc_rev table for the order.

The supplier number read in this function should be validated against the supplier number retrieved from the tables given the order number. This validation does not occur for vendor generated orders because the supplier number is not fetched in that case.

The dates retrieved are used for vendor generated orders or order date changes. Vendor generated orders and date changes require two dates, the not before date and the not after date.

Calls the function check_lock_ordhead.

validate_TITEM

This function processes the item data for purchase order acknowledgements. Item IDs are determined (ITEM, VPN & REF_ITEM). This function calls the process_item function to get required item and VPN information regarding order items.

This function also processes the pricing information for purchase order acknowledgements. If the order is vendor generated then no further processing is required. If it is a buyer generated order then further validation and program logic may be required. If the item is new and it has a single shipping location then the add_new_item function is called to add ordsku_rev and ordloc_rev data (If the order has multiple shipping locations then the add_new_item() function is called from the validate_TSHIP function.) New ordsku_rev and ordloc_rev records are added for the item/location and other ordloc_rev records are added via the add_new_loc function for further locations for the new item). If the item already exists, then the check_cost_chg function is called to determine the sent cost information differs from the current order cost information. If this is the case the insert_record_rev function will be called from the check_cost_chg function.

validate_TSHIP

This function processes the destination quantity line for purchase orders. After location and quantity location information is read, the add_new_item function will be called if the item is new to the order. Otherwise the check_qty_chg function is called to see if the sent item/location quantity differs from the item/location quantity currently held on ordloc for the order. If the quantities differ, the update_rev_qty function is called from the check_qty_chg function to update insert new information on the ordloc_rev table.

Calls the functions check_lock_eot and check_lock_olr.

cancel_order:

This function inserts into the ordhead_rev, ordloc_rev, and ordsku_rev tables. A zero quantity is written for the sku to be cancelled.

process_item

This function is called after the item information is read from the file. Sub-transaction level items should retrieve the transcaction level item number and VPN items (Vendor Catalog) should retrieve the transaction level item number for the VPN and supplier values in the EDI file. Buyer generated orders check to see if current item is on the ordsku table for the order. If the item does not exist, it is flagged as a new item.

Check_dates()

This function is called by validate_THEAD. The dates read from the input file are compared with the dates on the ordhead table. If either date has changed, the call the update_rev_dates() function to add a new revision with the modified not_before_date and not_after_date to the ordhead_rev table.

Check_cost_chg()

This function is called by validate_TITEM for existing items. The edi cost value read from the input file will be compared with order costs on the ordloc table. If any item/location has a different cost than the passed cost segment element then call the insert_record_rev() to adjust the cost amounts and cost_source on the ordloc_rev records.

Check_qty_chg()

This function is called the validate_TSHIP function. The sent quantity is compared with the quanitity stored on the ordloc table for the item/location. If the quantities differ or if the record does not exist (meaning the location does not exist for the order) then the update_rev_qty function is called to update/insert the proper quantities on the ordloc_rev table.

Update_rev_dates()

This function is called from the check_dates() function when passed a not_before_date and/or not_after_date that differ from the dates on the ordhead table. The function should insert a new record into ordhead_rev with the revised dates (if only one date has been revised, get the original value of the other date off the ordhead table and insert that value instead).

Insert_record_rev()

This function is called from the check_cost_chg() function when a passed cost amount differs from any item/location cost amount on the ordloc table. The function should insert the order/rev_no/item/location combination into the ordloc_rev table. The cost_source field should be set to 'MANL.'

Update_rev_qty

This function is called from the check_qty_chg() function when a passed qty amount for an item/location differs from the qty amount on the ordloc table for that order, item, and location. The function should update the revisions tables in the following manner:

- If the ordsku_rev entry has not been inserted for the current order_no/rev_no/origin_type/item combination, the function insert_ordsku_rev() will be called.
- If there is already an entry on ordsku_rev for the current order_no/rev_no/origin_type/item/location combination, the qty_ordered and qty_received will be updated for that entry. If there is no entry, a new one will be inserted.

Update_replenish()

This function is called to add records to the vendor replenishment tables. If the order is vendor generated then new records are inserted onto edi_ord_temp (or updated on this table if present). Information should be derived from the input file and the item tables.

Add_new_item()

This function inserts new records on the ordsku_rev table for an item new to the existing Oracle Retail buyer order. Insert new information to the ordsku_rev table with the following values: new revision number, 'R' (revision type), edi order number, blank case id, edi sku, edi upc, edi upc supplement, sku description from win_skus, null colour description, order_qty from input file, 0 for qty_received, cost source is 'MANL' if processed CTP segment for item, otherwise it's 'NRML', edi code type is null, prepack indicator is null, assortment indicator is null, and the rev_date is the vdate.

The add_new_loc function should be called to insert new ordloc_rev records.

Add_new_loc()

The function inserts records on the ordloc_rev tables for new locations on an order. It is called by the add_new_item function and from the update_rev_qty function. Determine whether the new location is a store or a warehouse. New information should be added

from the edi order. The cost is the passed cost if the cost was in the file for the item, otherwise it should be the item's unit_retail.

Get_location()

This function assigns values to store and warehouse variables depending on the value of the location value read from the edi input file. The store and warehouse tables should be hit to verify the location's existence.

Get_item_info

This function retrieves item information for inserts to edi_ord_temp and the revisions tables. It is called from add_new_item, add_new_loc. Item description, dept, and pack_ind is retrieved from the item_master table. Cost and retail are retrieved from the appropriate item tables.

get_item_info_wrap:

Calls get_item_info and also make_new_loc if necessary.

make_new_loc:

Calls the NEW_ITEM_LOC procedure to create a new item/location relationships.

get_pack_retail

Calls PRICING_ATTRIB_SQL.BUILD_PACK_RETAIL to get the unit retail price for a pack.

free_record_memory

Wipes clean the linked list containing all records on a transaction (stored in case the transaction needs to be rejected).

add_detail_record_to_list:

Adds a record to the linked lists

get_pack_ind

Gets the simple_pack_ind from packhead for a given pack.

check_lock_eot

Checks if the records to be updated in the edi_ord_temp table are locked.

check_lock_ordhead

Checks if the records to be updated in the ordhead table are locked.

check_lock_olr

Checks if the records to be updated in the ordloc_rev table are locked.

insert_batch_lock_log

Inserts the locked records in the batch_lock_log table.

cross_dock_po_validation

Validates the records of acknowledgement type VENDOR_ORDER (cross docked).

final:

Restart/recovery close; close input and reject files.

I/O Specification

Input file:

FHEAD file identification REQUIRED—one line per file
THEAD order and acknowledgment info REQUIRED—One line per acknowledgment
TITEM Item information OPTIONAL---multiple lines per acknowledgement possible
TSHIP location and quantity information OPTIONAL—multiple lines per item possible
TTAIL acknowledgment end marker REQUIRED—one line per acknowledgement
FTAIL file end marker REQUIRED—one line per file

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
	Oracle Retail file ID	Char(4)	ORAK	Describes file type
THEAD	Record descriptor	Char(5)	THEAD	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Acknowledge type	Char(2)		AP-product replenishment AK-no detail or change (use for UK receive, also) (NEW:CA-cancel order (no detail))
	Order number	Char(15)		May be external order number (vendor order number) OR Oracle Retail order number
	Written_date	Char(8)		Written date YYYYMMDD format
	Supplier number	Number(10)		Oracle Retail supplier number
	Not before date	Char(8)		Not_before_date YYYYMMDD
	Not after date	Char(8)		Not_after_date YYYYMMDD
	Purchase type	Char(6)		Specifies type of purchase – may be blank

Record Name	Field Name	Field Type	Default value	Description
	Pickup date	Char(8)		Pickup_date YYYYMMDD – may be blank
TITEM	Record descriptor	Char(5)	TITEM	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	ITEM	Char(25)		Item **At least one of Item, Ref_item
	Ref_item	Char(25)		Reference Item (for example UPC)
	Vendor catalog number	Char(30)		Vendor item number (VPN)
	Unit cost value	Number(20)		Unit_cost (4 implied decimal places)
	Loc_type	Char(2)		'ST' for store, 'WH' for warehouse
	Location	Number(10)		If NULL, apply to all locations for this item.
	Pickup location	Char(45)		Location to pick up item – may be blank
TSHIP	Record descriptor	Char(5)	TSHIP	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Store/wh indicator	Char(2)		'ST' for store, 'WH' for warehouse
	Ship to location	Number(10)		Store or warehouse number
	Quantity	Char(12)		Quantity ordered (4 implied decimal places)
TTAIL	Record descriptor	Char(5)	TTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Lines in transaction	Number(6)		Total number lines in this transaction
FTAIL	Record descriptor	Char(5)	FTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number(total # lines in file)

Record Name	Field Name	Field Type	Default value	Description
	Number of transactions	Number(10)		Number

vrplbld (Vendor Replenished Order Build)

Design Overview

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
EDI_ORD_TEMP	No	Yes	No	No	No
ORDHEAD	Yes	Yes	Yes	No	No
ORDSKU	No	No	Yes	Yes	No
ORDLOC	No	No	Yes	No	No
PERIOD	Yes	Yes	No	No	No
STORE	Yes	Yes	No	No	No
WH	Yes	Yes	No	No	No
UNIT_OPTIONS	No	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	Yes	No	No	No
SUPP_IMPORT_ATTR	Yes	Yes	No	No	No
SUPS	Yes	Yes	No	No	No
ORDSKU_HTS	Yes	Yes	Yes	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
DEAL_CALC_QUEUE	No	Yes	Yes	Yes	No
ORD_INV_MGMT	No	No	Yes	No	No

Note: ORDHEAD (select via index) is referenced via the function ORDER_NUMBER_SQL.NEXT_ORDER_NUMBER.

Indexes: ORDHEAD(order_no),
ORDSKU(order_no,item)

Function: To continue the process started by the program ediupack of building Oracle Retail orders that reflect the vendor-generated orders as received through the EDI 855.

The program ediupack.pc processes the EDI 855's that have been received from vendors. Rows have been inserted onto the table EDI_ORD_TEMP for use by this program. The EDI_ORD_TEMP table contains all items which were included on the EDI 855, along with the vendor order number with which they were originally associated.

The items are then consolidated so that one Oracle Retail order is created for each vendor-order-number/supplier.

Some details for each item are retrieved from the item/location tables and ITEM_SUPPLIER_COUNTRY and stored in ORDSKU and ORDLOC so that subsequent changes to the item do not affect the outstanding order.

A_order_amt is updated on the OTB table. If the supplier is foreign and OTB is calculated at cost, the added amount is total_duty + total_cost. If the supplier is not foreign and OTB is calculated at cost, the added amount is total_cost. Finally, if OTB is calculated at retail, the added amount is total_retail.

In records written to the ORDHEAD table:

- FREIGHT_TERMS is given the value fetched from SUPS.FREIGHT_TERMS
- ORIG_APPROVAL_DATE is assigned the value of the PERIOD.VDATE
- ORIG_APPROVAL_ID is assigned the value 'EDI855'

If an item is ordered for a location at which it does not have an item/location record, a record is built by calling the stored procedure NEW_ITEM_LOC.

For cross-dock VMI PO creation, it will allow having multiple shipping locations for same item/supplier combination. Among multiple shipping locations, one will be source warehouse and other can be store or warehouse, which will be sourced from source warehouse. The quantity given for source warehouse in TSHIP record should be greater than or equal to sum of quantities given in TSHIP record for cross docked locations.

The vendor-order-number will be used as the order number if it is a valid RMS pre-issued order number for the supplier. If the vendor-order-number is not a valid RMS pre-issued order number then the order number is allocated automatically on a 'next available' basis.

> **Note:** The vendor minimum order quantity is not considered when writing to the order tables because the quantity is determined by vendor. Thus, this quantity is not stored in the edi_ord_temp table.

Re-run: If this program aborts and terminates normally or terminates abnormally restart after setting restart_flag = 'Y' on restart_program_status table for current restart_name, schema, & thread_val.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	PHASE 3
Scheduling Diagram	N/A
Pre-Processing	N/A
Post-Processing	Vrplbld_post in prepost.pc truncates edi_ord_temp table
Threading Scheme	SUPPLIER

Restart Recovery

The logical unit of work (LUW) for vrplbld.pc will be a count of records pulled from the edi_temp_ord table. The number of records will be determined by the commit_max_ctr field on the restart_control table. Further, one or more of the key values (vendor_order_no, supplier, dept) must change before a commit event can take place.

This will ensure that only whole order will be saved, i.e. that a commit will never take place before all of the order detail records have been processed.

The commit_max_ctr field should be set to prevent excessive rollback space usage. Given the use of multiple threading, the recommended commit counter setting is 5000 records (subject to change based on experimentation).

Two cursors are defined, but only one cursor is opened for fetching (the criteria are based on whether or not departmental level ordering is allowed.)

```
EXEC SQL DECLARE c_items_d CURSOR FOR
    SELECT isc.origin_country_id,
           isc.supp_pack_size,
           NVL(isc.lead_time, 0),
           sia.agent,
           sia.lading_port,
           sia.discharge_port,
           s.currency_code,
           e.vendor_order_no,
           e.supplier,
           e.dept,
           e.wh_or_store_c,
           e.wh_or_store,
           e.item,
           e.item_desc,
           e.ref_item,
           e.unit_retail,
           DECODE(:edi_cost_override_ind, 'Y', iscl.unit_cost, e.unit_cost),
           e.qty_ordered,
           TO_CHAR(e.written_date,   'YYYYMMDD'),
           TO_CHAR(e.not_before_date,'YYYYMMDD'),
           TO_CHAR(e.not_after_date, 'YYYYMMDD'),
           TO_CHAR(e.not_after_date, 'DD'),
           TO_CHAR(e.not_after_date, 'MM'),
           TO_CHAR(e.not_after_date, 'YYYY'),
           ';' || e.vendor_order_no ||
           ';' || TO_CHAR(e.dept)   ||
           ';' || TO_CHAR(e.supplier)
      FROM item_supp_country isc,
           item_supp_country_loc iscl,
           sup_import_attr sia,
           sups s,
           edi_ord_temp e,
           v_restart_supplier rv
     WHERE isc.item               = e.item
       AND isc.primary_country_ind = 'Y'
       AND isc.supplier            = e.supplier
       AND sia.supplier (+)        = e.supplier
       AND s.supplier              = e.supplier
       AND rv.driver_value         = e.supplier
       AND rv.driver_name          = :os_restart_driver_name
       AND rv.num_threads          = :oi_restart_num_threads
       AND rv.thread_val           = :oi_restart_thread_val
       AND (e.vendor_order_no > NVL(:os_restart_vndr_ord_no,-999) OR
              (e.vendor_order_no = :os_restart_vndr_ord_no AND
                 (e.dept > :os_restart_dept OR
                    (e.dept = :os_restart_dept AND
                       e.supplier > :os_restart_supplier)
                 )
              )
           )
  ORDER BY e.vendor_order_no,
           e.dept,
           e.supplier,
```

```
            e.wh_or_store_c,
            e.wh_or_store,
            e.item;


EXEC SQL DECLARE c_items_s CURSOR FOR
    SELECT isc.origin_country_id,
           isc.supp_pack_size,
           NVL(isc.lead_time, 0),
           sia.agent,
           sia.lading_port,
           sia.discharge_port,
           s.currency_code,
           e.vendor_order_no,
           e.supplier,
           e.dept,
           e.wh_or_store_c,
           e.wh_or_store,
           e.item,
           e.item_desc,
           e.ref_item,
           e.unit_retail,
           DECODE(:edi_cost_override_ind, 'Y', iscl.unit_cost, e.unit_cost),
           e.qty_ordered,
           TO_CHAR(e.written_date,   'YYYYMMDD'),
           TO_CHAR(e.not_before_date,'YYYYMMDD'),
           TO_CHAR(e.not_after_date, 'YYYYMMDD'),
           TO_CHAR(e.not_after_date, 'DD'),
           TO_CHAR(e.not_after_date, 'MM'),
           TO_CHAR(e.not_after_date, 'YYYY'),
           ';' || e.vendor_order_no ||
           ';' || TO_CHAR(e.dept)   ||
           ';' || TO_CHAR(e.supplier)
      FROM item_supp_country isc,
           item_supp_country_loc iscl,
           sup_import_attr sia,
           sups s,
           edi_ord_temp e,
           v_restart_supplier rv
     WHERE isc.item               = e.item
       AND isc.primary_country_ind = 'Y'
       AND isc.supplier            = e.supplier
       AND sia.supplier (+)        = e.supplier
       AND s.supplier              = e.supplier
       AND rv.driver_value  = e.supplier
       AND rv.driver_name  = :os_restart_driver_name
       AND rv.num_threads  = :oi_restart_num_threads
       AND rv.thread_val   = :oi_restart_thread_val
       AND (e.vendor_order_no > NVL(:os_restart_vndr_ord_no,-999) OR
              (e.vendor_order_no = :os_restart_vndr_ord_no AND
                 e.supplier > :os_restart_supplier)
           )
  ORDER BY e.vendor_order_no,
           e.supplier,
           e.wh_or_store_c,
           e.wh_or_store,
           e.item;
```

Program Flow

N/A

Shared Modules

- NEW_ITEM_LOC
- ORDER_NUMBER_SQL.GET_NEXT_ORDER_NUMBER
- CURRENCY_SQL.GET_RATE
- CAL_TO_454_LDOW
- ORDER_SETUP_SQL.DEFAULT_ORDHEAD_DOCS
- ORDER_SETUP_SQL.DEFAULT_ORDSKU_DOCS
- ORDBA_CALC_SQL.ORDER_COST
- ORDER_EXPENSE_SQL.INSERT_COST_COMP
- ORDER_HTS_SQL.DEFAULT_CALC_HTS
- OTB_SQL.ORD_APPROVE
- ORDER_SETUP_SQL.DEFAULT_ORDER_INV_MGMT_INFO
- ORDER_ATTRIB_SQL.MULTIPLE_LOCS_EXIST
- SUP_INV_MGMT_SQL.GET_PURCHASE_PICKUP

Function Level Description

init()

- select department level orders flag from unit options
- select base country, latest ship days, FOB information, ELC indicator, EDI cost override inidicator, calendar 454 indicator, bill-to location, and multi-channel indicator from system options
- select vdate from period
- call restart initialization logic

process()

- call open_cursor to open appropriate cursor (department level or multi-department orders)
- call fetch_cursor for prime fetch of driving cursor
- while
- if supplier or vendor order number changed then
- call create_header
  - if item changed
    - call create_item
- if multi-channel = 'N' or the location fetched is a store
- call add_loc
- If multi-channel is on and the location is a warehouse, then distribute:
- call dist_init
- call dist_set_distribution_rule
- for every replenishable warehouse
- call dist_expected_quantity
- call dist_distributed_quantity
- call add_loc
- call dist_final

  - call add_loc
  - call fetch_cursor
- if supplier or vendor order number changed then
  - if the ELC indicator is Y, call add_cost_comp
    - call item_defaults
  - call apply_disc
  - call header_defaults
  - call update_header
  - call update_order
  - call check_vmi_order
- else if the item changed
  - if the ELC indicator is Y, call add_cost_comp
    - call item_defaults
  - call restart commit logic
- end while
  - call item_defaults
  - call apply_disc
- call header_defaults
  - call update_header
- call update_order
  - call check_vmi_order

open _cursor()

open appropriate cursor depending on what type of orders are allowed.

fetch _cursor()

fetch records from appropriate cursor

get_min_max()

finds the minimum and maximum lead times for the current order

item_active ()

returns 1 if item is active (0 if inactive) on item_loc

create_header()

- call function to determine if vendor-order-number is a valid pre-issued order number (ORDER_NUMBER_SQL.CHECK_ORDER_PREISSUE)
- If not pre-issued order number call function to get the next order number (ORDER_NUMBER_SQL.NEXT_ORDER_NUMBER)
- call CAL_TO_454_LDOW procedure to convert order's not_after_date to the end-of-week date to update otb_eow_date field
- call CURRENCY_SQL.GET_RATE to get the exchange rate for the supplier's currency.
- insert record into ordhead

header_defaults()

- Call ORDER_ATTRIB_SQL.MULTIPLE_LOCS_EXIST to retrieve the location if the order is a single location order. Call ORDER_SETUP_SQL.DEFAULT_ORDHEAD_DOCS to default the necessary documents for the order header. Also call ORDER_SETUP_SQL.
- DEFAULT_ORDER_INV_MGMT to appropriate supplier inventory management information to the order with NULL inserted for the scaling constraints, since a vendor generated order is not eligible for scaling. If no values are found for the purchase type or pickup location, call SUP_INV_MGMT_SQL.GET_PURCHASE_PICKUP to retrieve the default values. If no value is found for the pickup date, use the longest lead-time to calculate the pickup date.

update_header()

Check to see if any items on the order come from a country other than the ordhead import country. If so, update the import_order_ind to 'Y' on ordhead.

update_order()

Update the Order Header table with the following values

- Purchase Type
- Pickup Location (only enter if the Purchase Type is 'FOB' or 'BACK')
- Pickup Date (only enter if the Purchase Type is 'FOB' or 'BACK') – if value is later than the Not After Date, set equal to the Not After Date
- Earliest Ship Date – minimum of Order/Item ESDs
- Latest Ship Date – maximum of Order/Item LSDs

Update Order Item table. Set the pickup location if the purchase type is 'FOB' or 'BACK'.

item_defaults()

Call ORDER_SETUP_SQL.DEFAULT_ORDSKU_DOCS to default necessary documents for items in order.

apply_disc()

if edi_cost_override_ind = 'Y' then insert a record into the DEAL_CALC_QUEUE table if the current order number is not in the table, or update the order_appr_ind field to 'Y' if the order number is already in the table.

call update_OTB

add_cost_comp()

- Call ORDER_EXPENSE_SQL.INSERT_COST_COMP,
- ORDER_HTS_SQL.DEFAULT_CALC_HTS and
- ELC_CALC_SQL.CALC_COMP.

create_item ()

if edi_cost_override_ind = 'Y' insert record into ordsku with the cost_source = 'MANL'. If edi_cost_override_ind != 'Y', insert a record into ordsku with the cost_source = 'NORM';

add_loc ()

insert into ordloc

set_worksheet_status()

For a given order number and reject code, update the status on the ordhead table to 'W'orksheet and update the reject code to whatever was passed in.

order_exists()

For a given order number, check to see whether or not the order number is on the DEAL_CALC_QUEUE table.

final()

Call restart/recovery close logic.

I/O Specification

N/A