Oracle® Retail Merchandising System

Operations Guide, Volume 2 - Message Publication and Subscription Design

Release 14.1.3

E83764-01

February 2017

ORACLE

Oracle® Retail Merchandising System Operations Guide, Volume 2 - Message Publication and Subscription Design, Release 14.1.3

E83764-01

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Primary Author: Maria Andrew

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

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 **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.

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

(iii) 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.

(iv) 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.

You acknowledge and confirm that Oracle grants you use of only the object code of the VAR Applications. Oracle will not deliver source code to the VAR Applications to you. Notwithstanding any other term or condition of the agreement and this ordering document, you shall not cause or permit alteration of any VAR Applications. For purposes of this section, "alteration" refers to all alterations, translations, upgrades, enhancements, customizations or modifications of all or any portion of the VAR Applications including all reconfigurations, reassembly or reverse assembly, re-engineering or reverse engineering and recompilations or reverse compilations of the VAR Applications or any derivatives of the VAR Applications. You acknowledge that it shall be a breach of the agreement to utilize the relationship, and/or confidential information of the VAR Applications for purposes of competitive discovery.

The VAR Applications contain trade secrets of Oracle and Oracle's licensors and Customer shall not attempt, cause, or permit the alteration, decompilation, reverse engineering, disassembly or other reduction of the VAR Applications to a human perceivable form. Oracle reserves the right to replace, with functional equivalent software, any of the VAR Applications in future releases of the applicable program.

Contents

Send Us Your Comments

Oracle Retail Merchandising System Operations Guide, Volume 2 - Message Publication and Subscription Design, Release 14.1.3

Oracle welcomes customers' comments and suggestions on the quality and usefulness of this document.

Your feedback is important, and helps us to best meet your needs as a user of our products. For example:

- Are the implementation steps correct and complete?
- Did you understand the context of the procedures?
- Did you find any errors in the information?
- Does the structure of the information help you with your tasks?
- Do you need different information or graphics? If so, where, and in what format?
- Are the examples correct? Do you need more examples?

If you find any errors or have any other suggestions for improvement, then please tell us your name, the name of the company who has licensed our products, the title and part number of the documentation and the chapter, section, and page number (if available).

> **Note:** Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the new Applications Release Online Documentation CD available on My Oracle Support and www.oracle.com. It contains the most current Documentation Library plus all documents revised or released recently.

Send your comments to us using the electronic mail address: retail-doc_us@oracle.com

Please give your name, address, electronic mail address, and telephone number (optional).

If you need assistance with Oracle software, then please contact your support representative or Oracle Support Services.

If you require training or instruction in using Oracle software, then please contact your Oracle local office and inquire about our Oracle University offerings. A list of Oracle offices is available on our Web site at www.oracle.com.

Preface

The *Oracle Retail Merchandising System Operations Guide, Volume 2 - Message Publication and Subscription Design* provides critical information about the processing and operating details of Oracle Retail Merchandising System (RMS), including the following:

- System configuration settings
- Technical architecture
- Functional integration dataflow across the enterprise
- Batch processing

Audience

This guide is for:

- Systems administration and operations personnel
- Systems analysts
- Integrators and implementers
- Business analysts who need information about Merchandising System processes and interfaces

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following documents:

- *Oracle Retail Merchandising System Installation Guide*
- *Oracle Retail Merchandising System Release Notes*
- *Oracle Retail Merchandising System Data Model*
- *Oracle Retail Merchandising System Data Access Schema Data Model*
- *Oracle Retail Merchandising Batch Schedule*
- Oracle Retail Sales Audit documentation
- Oracle Retail Trade Management documentation
- Oracle Retail Fiscal Management documentation

Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:
https://support.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

When you install the application for the first time, you install either a base release (for example, 14.1) or a later patch release (for example, 14.1.3). If you are installing the base release or additional patch releases, read the documentation for all releases that have occurred since the base release before you begin installation. Documentation for patch releases can contain critical information related to the base release, as well as information about code changes since the base release.

Improved Process for Oracle Retail Documentation Corrections

To more quickly address critical corrections to Oracle Retail documentation content, Oracle Retail documentation may be republished whenever a critical correction is needed. For critical corrections, the republication of an Oracle Retail document may at times **not** be attached to a numbered software release; instead, the Oracle Retail document will simply be replaced on the Oracle Technology Network Web site, or, in the case of Data Models, to the applicable My Oracle Support Documentation container where they reside.

This process will prevent delays in making critical corrections available to customers. For the customer, it means that before you begin installation, you must verify that you have the most recent version of the Oracle Retail documentation set. Oracle Retail documentation is available on the Oracle Technology Network at the following URL:

`http://www.oracle.com/technetwork/documentation/oracle-retail-100266.html`

An updated version of the applicable Oracle Retail document is indicated by Oracle part number, as well as print date (month and year). An updated version uses the same part number, with a higher-numbered suffix. For example, part number E123456-**02** is an updated version of a document with part number E123456-**01**.

If a more recent version of a document is available, that version supersedes all previous versions.

Oracle Retail Documentation on the Oracle Technology Network

Oracle Retail product documentation is available on the following web site:

http://www.oracle.com/technetwork/documentation/oracle-retail-100266.html

(Data Model documents are not available through Oracle Technology Network. You can obtain them through My Oracle Support.)

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".

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

1

Introduction

RMS Operations Guide, Volume 2 - Message Publication and Subscription Design contains detailed technical information about how RMS interacts with the Oracle Retail Integration Bus (RIB). The major components of this volume of *Retail Merchandising System Operations Guide, Volume 2 - Message Publication and Subscription Design* are described in the preceding sections.

Message Publication and Subscription Designs

Oracle Retail Integration Bus (RIB) RMS functional overview are incorporated into the publication and subscription designs. Therefore, the retailer can extract the business rationale behind each publication or subscription as well as the technical details that describe, on a technical level, how RMS publishes messages to the RIB or how RMS subscribes to the message from the RIB.

External Subscription RIB Application Programming Interface

Subscription Application Programming Interface (API) that is designated as External is designed to be interfaces for external systems that maintain the applicable data. In other words, RMS is not the System Of Record for maintaining the data. Instead, RMS subscribes to consume the data when it is published so that the corresponding data in RMS can be kept in sync with the external system that maintains the data.

Parallel processing for Performance Purpose

Parallel processing threading capability for a message family is limited by the parallel processing support in the publishing performed by applications. For example, the Inventory Adjustment (InvAdjust) message family is published by the Oracle Retail Warehouse Management System (RWMS) and subscribed by RMS. Because RWMS supports only single process publishing, RMS needs to be set up for single process subscription for the InvAdjust message family.

The majority of publishing and all of the subscribing APIs support parallel processing. The APIs that do and do not support parallel processing publication are listed in the following:

Subscription APIs

- All RMS subscription APIs support parallel processing.

Publishing APIs

The following RMS publishing APIs support parallel processing:

- RMSMFM_ALLOCB (Allocations Publication API)
- RMSMFM_ITEMLOCB (Item Location Publication API)
- RMSMFM_ITEMSB (Item Publication API)
- RMSMFM_MERCHHIERB (Merchandise Hierarchy Publishing API)
- RMSMFM_ORDERB (Order Publication API)
- RMSMFM_RCVUNITADJB (Receiver Unit Adjustment Publication API)
- RMSMFM_RTVREQB (RTV Request Publication API)
- RMSMFM_SHIPMENTB (ASNOUT Publication API)
- RMSMFM_TRANSFERSB (Transfers Publication API)
- RMSMFM_WOINB (Work Orders in Publication API)
- RMSMFM_WOOUTB (Work Orders out Publication API)

The following RMS publishing APIs *do not* support parallel processing:

- RMSMFM_BANNERB (Banner Publication API)
- RMSMFM_DIFFGRPB (Differentiator Groups Publication API)
- RMSMFM_DIFFIDB (Differentiator ID Publication API)
- RMSMFM_DLVYSLTB (Delivery Slot Publication API)
- RMSMFM_PARTNERB (Partner Publication API)
- RMSMFM_SEEDDATAB (Seed Data Publication API)
- RMSMFM_SEEDOBJB (Seed Object Publication API)
- RMSMFM_STOREB (Store Publication API)
- RMSMFM_SUPPLIERB
- RMSMFM_UDAB (UDA Publication API)
- RMSMFM_WHB (Warehouse Publication API)

Service Provider Implementations API Designs

The Service Provider Implementations API Designs chapter provides a high level overview of the APIs. The implementation of these services, along with the associated Web Service Definition Language (WSDL), may be used to get a full understanding of the data requirements, validation rules, persistence rules, and return values associated with the service.

2

Publication Designs

This chapter provides an overview of the Publication APIs used in the RMS environment and various functional attributes used in the APIs.

For more information on RIB_XML, see Appendix: RIB_XML.

Allocations Publication API

Functional Area

Allocations

Business Overview

RMS is responsible for communicating allocation information with external systems such as Oracle Retail Store Inventory Management (SIM) or Oracle Retail Warehouse Management System (RWMS).

Allocation data enters RMS through the following ways:

- Through the Oracle Retail Allocation product.
  These allocations are written to the ALLOC_HEADER and ALLOC_DETAIL tables in 'R'- Reserved or 'A'- Approved status. Once a detail and a header message have been queued and approved, a message is published to the RIB.
- Through the semi-automatic ordering option.
  Using this replenishment method, allocations and orders are inserted into the ALLOC_HEADER and ALLOC_DETAIL tables in worksheet status to be manually approved. In order for allocation messages to be published to the RIB, the allocation must at least be in approved status. Worksheet messages remain on the queue and combined until they are approved. When it is approved, the created message is published to the RIB.
- Through automatic replenishment allocations.
  These allocations are initially set in worksheet status and are approved by the RPLAPPRV.PC batch program (Replenishment Approve). Only messages for approved allocations are published to the RIB.
- Through the Allocation subscription RIB API.
- Either a 3rd party Merchandise System or AIP can create allocations in RMS. Once approved, these allocations are published to the RIB.

Allocations can be created from a warehouse to any type of stockholding location in RMS, including both company and franchise stores. Allocations include a store type and stockholding indicator at the detail level when allocating to stores, to allow SIM and RWMS to filter out the data irrelevant to their respective systems. When allocating to a franchise store, the linked franchise orders are not published; only the allocation itself is published.

An allocation and its details are not published until it is approved. Modified and deleted allocation information is also sent to the RIB. Allocation header modification messages will be sent if the status of the allocation is changed to 'C' - closed or if the allocation release date is changed. Allocation detail modification messages will be sent if the

allocated quantity is changed. A header delete message signifies that the complete allocation can be deleted.

Package Impact

Business Object ID

Allocation number

Create Header

1. **Prerequisites**: Allocation can be created in one of the following manners: via the stand-alone allocations product, semi-automatic ordering, automatic ordering replenishment, or Allocation subscription API.
2. **Activity Detail**: Once an allocation exists in RMS it can be modified and details can be attached.
3. **Messages**: When an allocation is created an Allocation Create message request is queued. The Allocation Create message is a flat message containing a full snapshot of the allocation at the time the message is published. The message will not be sent until detail records have been queued and the allocation has been approved.

Modify Header

1. **Prerequisites**: An allocation must exist before it can be modified.
2. **Activity Detail**: The user is allowed to change the status of the allocation to 'A'- Approved or 'C'- Closed. This change is of interest to other systems and so this activity results in the publication a message.
3. **Messages**: When an allocation is modified, an Allocation Header Modified message request is queued. The Allocation Header Modified message is a flat message containing a full snapshot of the allocation header at the time the message is published.

Create Detail

1. **Prerequisites**: An allocation header must exist before an allocation detail can be created or interfaced into RMS. Once in RMS, the allocation can only be modified by changing its allocated quantity.
2. **Activity Detail**: an Allocation Detail Create message is only queued if a Create Header message is also on the queue for the same allocation.
3. **Messages**: When an allocation detail is created, an Allocation Detail Created message request is queued. The Allocation Detail Create message is a flat message containing a full snapshot of the allocation detail at the time the message is published. If an Allocation Create message is also in the queue for the same allocation, the two messages are combined and sent as one message.

Modify Detail

1. **Prerequisites**: An allocation detail must exist to be modified.
2. **Activity Detail**: The user is allowed to change allocation quantities provided they are not reduced below those already recorded as received. This change is of interest to other systems and so this activity results in the publication of a message.
3. **Messages**: When an allocation is modified an Allocation Detail Modified message request is queued. The Allocation Detail Modified message is a flat message containing a full snapshot of the allocation detail at the time the message is published.

Approve

1. **Prerequisites**: An allocation must exist in RMS before it can be approved. Those allocations created from other sources can be entered into RMS in approved status.
2. **Activity Detail**: Once an allocation as been approved, it it will be published from RMS.
3. **Messages**: When the allocation is approved an Allocation Header Modified message is queued. This message will be combined with any Allocation Create and Allocation Detail Create message to form the message that is sent to the RIB.

Close

1. **Prerequisites**: An allocation must be approved before it can be closed.
2. **Activity Detail**: Closing an allocation changes the status, which prevents further receiving or modification of the allocation. When an allocation is closed, a message is published to update other systems regarding the status change.
3. **Messages**: Closing an allocation queues an Allocation Header Modified message request. This is a flat message containing a full snapshot of the allocation at the time that the message is published.

Delete

1. **Prerequisites**: An allocation can only be deleted when it is still in approved status or when it has been closed.

   Note: That if the allocation is in closed status, it still cannot be deleted if either create or a modify message are pending for the allocation, as they need to take full snapshots.

2. **Activity Detail**: Deleting an allocation removes it from the system. External systems are notified by a published message.
3. **Message**: When an allocation is deleted, an Allocation Header Deleted message, which is a flat notification message, is queued.

Package Name: RMSMFM_ALLOC

Body File Name: rmsmfm_allocb.pls

Functional Level Description – ADDTOQ

```
FUNCTION ADDTOQ (  O_error_msg       OUT     VARCHAR2,
           I_message_type            IN      ALLOC_MFQUEUE.MESSAGE_TYPE%TYPE,
           I_alloc_no                IN      ALLOC_HEADER.ALLOC_NO%TYPE,
           I_alloc_header_status     IN      ALLOC_HEADER.STATUS%TYPE,
           I_to_loc                  IN      ITEM_LOC.LOC%TYPE)
```

This function is called by the ALLOC_HEADER trigger and the ALLOC_DETAIL trigger, ec_table_alh_aiudr and ec_table_ald_aiudr, respectively.

- For header level insert messages (HDR_ADD), insert a record in the ALLOC_PUB_INFO table. The published flag will be set to 'N'. The correct thread for the business transaction will be calculated and written. Call API_LIBRARY.RIB_SETTINGS to get the number of threads used for the publisher. The number of threads and the business object ID are used to calculate the thread value.
- For all records except header level inserts (HDR_ADD), the thread_no and initial_approval_ind will be queried from the ALLOC_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 will take place and the function will exit.
- For detail level message deletes (DTL_DEL), we only need one (the most recent) record per detail in the ALLOC_MFQUEUE. Delete any previous records that exist on the ALLOC_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 ALLOC_MFQUEUE. Delete any previous DTL_UPD records that exist on the ALLOC_MFQUEUE for the record that has been passed.
- For header level delete messages (HDR_DEL), delete every record in the queue for that allocation.
- For header level update message (HDR_UPD), update the ALLOC_PUB_INFO.INITIAL_APPROVAL_IND to 'Y' if the allocation is in approved status.
- For all records except header level inserts (HDR_ADD), insert a record into the ALLOC_MFQUEUE.

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

Functional 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 ALLOC_MFQUEUE table (PUB_STATUS = 'U'). It will only need to execute one loop iteration in most cases. For each record retrieved, GETNXT does the following:

- 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.
- If the lock is successful, a check for records on the queue with a status of 'H'-Hospital. If there are any such records for the current business object, GETNXT raises an exception to send the current message to the Hospital.
- The information from the ALLOC_MFQUEUE and ALLOC_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

This procedure republishes the entity that failed to be published before. It is the same as GETNXT except that the record on ALLOC_MFQUEUE to be published must match the passed in sequence number contained in the ROUTING_INFO.

Function Level Description – PROCESS_QUEUE_RECORD (local)

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.

- Call API_LIBRARY.RIB_SETTINGS to get the number of threads used for the publisher. The number of threads and the business object ID are used to calculate the thread value
- For a header delete message (HDR_DEL) that has not been initially published, simply remove the header delete message from the queue and loop again.
- For a header delete message (HDR_DEL) that has been initially published i.e. for AllocRef.
- Build the Oracle Object to publish to RIB.
- Build the ROUTING_INFO.
- Delete the record from ALLOC_PUB_INFO.
- Delete the record from ALLOC_DETAILS_PUBLISHED.
- Remove the header delete message from the queue (ALLOC_MFQUEUE).
- If the business object is being published for the first time i.e. published_ind on the pub_info table is 'N', the business object is being published for the first time. If so, call MAKE_CREATE.
- Otherwise, For a header update message (HDR_UPD).
- Call BUILD_HEADER_OBJECT to build the Oracle Object to publish to the RIB.
- Update ALLOC_PUB_INFO with updated new header information.
- Build the ROUTING_INFO.
- Delete the header update message from the queue (ALLOC_MFQUEUE).
- For a detail add (DTL_ADD) or detail update message (DTL_UPD).
- Call BUILD_DETAIL_CHANGE_OBJECTS to build the Oracle Object to publish to the RIB. This will also take care of any ALLOC_MFQUEUE deletes and ROUTING_INFO logic.
- For a detail delete message (DTL_DEL).
- Call BUILD_DETAIL_DELETE_OBJECTS to build the Oracle Object to publish to the RIB. This will also take care of any ALLOC_MFQUEUE and ALLOC_DETAILS_PUBLISHED deletes and the ROUTING_INFO logic.

Function Level Description – MAKE_CREATE (local)

This function is used to create the Oracle Object for the initial publication of a Business transaction.

- Call BUILD_HEADER_OBJECT to get a header level Oracle Object plus any extra functional holders.
- Build some or all of the ROUTING_INFO Oracle Object.
- Call BUILD_DETAIL_OBJECTS to get a table of detail level Oracle objects and a table of ALLOC_MFQUEUE rowids to delete.
- Use the header level Oracle Object and functional holders to update the ALLOC_PUB_INFO.
- Delete records from the ALLOC_MFQUEUE for all rowids returned by BUILD_DETAIL_OBJECTS. Deletes are done by rowids instead of business transaction keys to ensure that nothing is deleted off the queue that has not been published.
- If the entire business transaction was added to the Oracle Object, also delete the ALLOC_MFQUEUE record that was picked up by GETNXT. If the entire business transaction was not published, we need to leave something on the ALLOC_MFQUEUE to ensure that the rest of it is picked up by the next call to GETNXT.
- The header and detail level Oracle Objects are combined and returned.

Function Level Description – BUILD_HEADER_OBJECT (local)

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

Optionally can return needed Functional Holders for the ALLOC_PUB_INFO table.

The C_ALLOC_HEAD cursor selects the context fields (context and value) from the ALLOC_HEADER table.

The context fields will be passed along in the parameter list of the rib object constructor "RIB_AllocDesc_REC()".

Function Level Description – BUILD_DETAIL_OBJECTS (local)

The function is responsible for building detail level Oracle Objects. It builds as many detail Oracle Object as it can given the passed in message type and business object keys.

If the function is being called from MAKE_CREATE:

Select any unpublished detail records from the business transaction (use an indicator on the functional detail table itself or ALLOC_DETAILS_PUBLISHED). Create Oracle Objects for details that are selected by calling BUILD_SINGLE_DETAIL.

- Ensure that the indicator in the functional detail table is updated as published as the detail info are placed into the Oracle Objects
- Ensure that ALLOC_MFQUEUE is deleted as needed. If there is more than one ALLOC_MFQUEUE record for a detail level record, make sure they all get deleted. We only care about current state, not every change.
- Ensure that ROUTING_INFO is constructed if routing information is stored at the detail level in the Business transaction.
- Ensure that no more than MAX_DETAILS_TO_PUBLISH records are put into Oracle Objects.
- Ensure that the detail records being added to the object have not already been published. This can happen if GETNXT was previously called for the current

business object, and the MAX_DETAILS_TO_PUBLISH limit had been reached. We ensure these details do not get added again by looking at the indicator in the functional detail table.

If the function is not being called from MAKE_CREATE:

Select any details on the ALLOC_MFQUEUE that are for the same business transaction and for the same message type. Create Oracle Objects for details that are selected by calling BUILD_SINGLE_DETAIL.

- If the message type is a detail create (DTL_ADD), ensure that records get inserted into ALLOC_DETAILS_PUBLISHED or the indicator in the functional detail table is updated as published because the detail info are placed into the Oracle Objects.
- Ensure that ALLOC_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.
- Ensure that no more than MAX_DETAILS_TO_PUBLISH records are put into Oracle Objects.

The deletes are done by ROWID to make sure that records from the queue table that has not been published are not deleted.

Function Level Description – BUILD_SINGLE_DETAIL (local)

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

Function Level Description – BUILD_DETAIL_CHANGE_OBJECTS (local)

Either pass in a header level Oracle Object or call BUILD_HEADER_OBJECT to build one.

Call BUILD_DETAIL_OBJECTS to get the detail level Oracle Objects.

Perform any BULK DML statements given the output from BUILD_ DETAIL_OBJECTS and update to ALLOC_DETAILS_PUBLISHED.

Build any ROUTING_INFO as needed.

Function Level Description – BUILD_DETAIL_DELETE_OBJECTS (local)

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

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

Perform any BULK DML statements for deletion from ALLOC_MFQUEUE and update to ALLOC_DETAILS_PUBLISHED.

Build any ROUTING_INFO as needed.

Function Level Description – LOCK_THE_BLOCK (local)

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 the queue table for DTL_UPD, DTL_DEL, and HDR_DEL messages.

Function Level Description – HANDLE_ERRORS (local)

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 ALLOC_MFQUEUE record back to the RIB in the ROUTING_INFO. It sends back a status of 'H' - Hospital to the RIB as well. It then updates the status of the queue record to 'H', 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' - Error 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' to 'E'.

Function Level Description – DELETE_QUEUE_REC (local)

This function deletes a specific record on ALLOC_MFQUEUE table depending on the seq_no.

Function Level Description – GET_ROUTING_TO_LOCS (local)

This function will get all the values of to_loc_vir from alloc_details_published table depending on a given allocation number.

Perform a cursor for loop that will populate the Oracle Object RIB_ROUTINGINFO_TBL.

Function Level Description – GET_NOT_BEFORE_DAYS (local)

This function checks if the variable (LP_nbf_days) has a value or not. If not, it will populate the variable based on code_detail and then assign this value to the variable O_days.

Function Level Description – GET_RETAIL (local)

This function will accept inputs and pass it to PRICING_ATTRIB_SQL.GET_RETAIL function to get the retail value of the item.

Function Level Description – CHECK_STATUS (local)

CHECK_STATUS raises an exception if the status code is set to 'E' - Error. This will be called immediately after calling a procedure that sets the status code. Any procedure that calls CHECK_STATUS must have its own exception handling section.

Trigger Impact

Trigger name: EC_TABLE_ALH_AIUDR

Trigger file name: ec_table_alh_aiudr.trg

Table: ALLOC_HEADER

- **Inserts:** Send the allocation header level information to the ADDTOQ procedure in RMSMFM_ALLOC with the message type RMSMFM_ALLOC.HDR_ADD and the original message.
- **Updates:** Send the allocation header level information to the ADDTOQ procedure in the RMSMFM_ALLOC with the message type RMSMFM_ALLOC.HDR_UPD and the original message.
- **Deletes:** Send the allocation header level info to the ADDTOQ procedure in the RMSMFM_ALLOC with the message type RMSMFM_ALLOC.HDR_DEL and the original message.

Trigger name: EC_TABLE_ALD_AIUDR

Trigger file name: ec_table_ald_aiudr.trg

Table: ALLOC_DETAIL

- **Inserts:** Send the allocation detail level information to the ADDTOQ procedure in RMSMFM_ALLOC with the message type RMSMFM_ALLOC.DTL_ADD and the original message.
- **Updates:** Send the allocation detail level information to the ADDTOQ procedure in the RMSMFM_ALLOC with the message type RMSMFM_ALLOC.DTL_UPD and the original message.
- **Deletes:** Send the allocation detail level info to the ADDTOQ procedure in the RMSMFM_ALLOC with the message type RMSMFM_ALLOC.DTL_DEL and the original message.

Message XSD

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	XML Schema Definition (XSD)
AllocCre	Allocation Create Message	AllocDesc.xsd
AllocHdrMod	Allocation Header Modify Message	AllocDesc.xsd
AllocDel	Allocation Delete Message	AllocRef.xsd
AllocDtlCre	Allocation Detail Create Message	AllocDesc.xsd
AllocDtlMod	Allocation Detail Modify Message	AllocDesc.xsd
AllocDtlDel	Allocation Detail Delete Message	AllocRef.xsd

Design Assumptions

None

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ALLOC_PUB_INFO	Yes	Yes	Yes	No
ALLOC_MFQUEUE	Yes	Yes	No	Yes
ALLOC_DETAILS_PUBLISHED	Yes	Yes	Yes	Yes
ALLOC_HEADER	Yes	No	No	No
ALLOC_DETAIL	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
ITEM_TICKET	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
WH	Yes	No	No	No

TABLE	SELECT	INSERT	UPDATE	DELETE
ORDHEAD	Yes	No	No	No
CODE_DETAIL	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No

ASNOUT Publication API

Functional Area

ASNOut

Business Overview

ASNOUT means the outbound message of Advanced Shipment Notification. The ASN out message is used to ship the merchandise against transfers or allocations. This message is published by RMS to stores or warehouses.

RMS supports the following shipping functionality:

- On-line Shipping/Receiving.
- Franchise Order Shipment and Return.

On-line Shipping/Receiving

Two system options (ship_rcv_store and ship_rcv_wh) are used to control whether RMS on-line shipment/receiving functionality is enabled.

- Ship_rcv_store = 'Y' means a store inventory management application, such as Oracle Retail SIM, is NOT installed and shipping/receiving for stores will be done in RMS.
- Ship_rcv_wh = 'Y' means a warehouse management system, such as RWMS, is NOT installed and shipping/receiving for warehouses will be done in RMS.

If either (but not both) of these indicators is set to 'Y', shipments created in RMS should be published to the RIB to allow the integration subsystem application to have visibility to the corporately created shipment.

The possible scenarios for on-line shipping/receiving:

SIM Installed (Yes/No)	RWMS Installed (Yes/No)	System Options Settings	RMS Publishes Shipments (Yes/No)	Apps to subscribe to the message (SIM/RWMS)
Yes	Yes	Ship_rcv_store = N Ship_rcv_wh = N	No	No
No	No	Ship_rcv_store = Y Ship_rcv_wh = Y	No	No
Yes	No	Ship_rcv_store = N Ship_rcv_wh = Y	Yes – for warehouse-to-store shipments	SIM
No	Yes	Ship_rcv_store = Y Ship_rcv_wh = N	Yes – for store-to-warehouse shipments	RWMS

RMS on-line shipping can involve a customer order transfer (tsf_type = 'CO'). For a customer order transfer, customer order number, and fulfillment order number are pulled from the ORDCUST table and included in the published information.

Franchise Order Shipment and Return

Franchise stores are a special kind of stores that are not 'owned' by the company; therefore any shipment to a franchise store is considered a sale. From RMS, franchise stores can order goods from company stores or warehouses; they can also return goods back to company stores or warehouses. These orders and returns are created as transfers in RMS.

RMS supports two kinds of franchise stores – stockholding franchise stores (which RMS manages inventory and financials like regular stores) and non-stockholding franchise stores (which RMS does NOT manage inventory and financials).

SIM manages transactions for stockholding franchise stores, but not for non-stockholding franchise stores. The Shipping and Receiving of non-stockholding franchise orders and returns are handled within RMS from the Store perspective even if SIM is installed.

For warehouses, if a franchise return from a non-stockholding franchise store is to be processed, RWMS will require an ASN against which to receive. Since RMS automatically creates the shipment for non-stockholding stores upon the approval of a franchise return, RMS needs to publish those shipments for RWMS. Similar to on-line Shipping/Receiving, RMS publishes shipments of non-stockholding Franchise Returns to warehouses as ASNOut messages.

Package Impact

Business Object ID

Shipment number

Package name: RMSMFM_SHIPMENT

Function Level Description – ADDTOQ

```
ADDTOQ (O_error_message   IN OUT   RTK_ERRORS.RTK_TEXT%TYPE,
        I_message_type    IN       SHIPMENT_PUB_INFO.MESSAGE_TYPE%TYPE,
        I_shipment        IN       SHIPMENT.SHIPMENT%TYPE,
        I_to_loc          IN       SHIPMENT.TO_LOC%TYPE,
        I_to_loc_type     IN       SHIPMENT.TO_LOC_TYPE%TYPE)
```

- Shipments created in RMS cannot be modified. Upon saving a shipment, the entire shipment is published from RMS as one ASNOut message. As a result, RMS only needs to support the ASNOut create message type ('asnoutcre') for shipment publishing.
- Validate all the input parameters to this function against NULL. If any has a NULL value then return from the function with the appropriate error message.
- Insert a record in the SHIPMENT_PUB_INFO table. The published flag will be set to 'U'. The correct thread for the business transaction will be calculated and written. Call API_LIBRARY. GET_RIB_SETTINGS to get the number of threads used for the publisher. Using the number of threads, and the business object ID (For example, shipment number), calculate the thread value.

Function Level Description – GETNXT

```
GETNXT (O_status_code     IN OUT   VARCHAR2,
        O_error_message   IN OUT   RTK_ERRORS.RTK_TEXT%TYPE,
        O_message_type    IN OUT   VARCHAR2,
        O_message         IN OUT   RIB_OBJECT,
        O_bus_obj_id      IN OUT   RIB_BUSOBJID_TBL,
        O_routing_info    IN OUT   RIB_ROUTINGINFO_TBL,
        I_num_threads     IN       NUMBER DEFAULT 1,
        I_thread_val      IN       NUMBER DEFAULT 1)
```

Initialize LP_error_status to API_CODES.HOSPITAL at the beginning of GETNXT.

The RIB calls GETNXT to get messages. It performs a cursor loop on the unpublished records on the SHIPMENT_PUB_INFO table (PUB_STATUS = 'U'). It will only 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 objects (i.e. shipment number). 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. A check for records on the queue with a status of 'H' -Hospital. If there are any such records for the current business object, GETNXT raises an exception to send the current message to the Hospital.
3. The information from the SHIPMENT_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.
4. If any exception is raised in GETNXT, including the exception raised by an unsuccessful call to PROCESS_QUEUE_RECORD, HANDLE_ERRORS is called.
5. Unconditionally exit from the loop after the successful processing of PROCESS_QUEUE_RECORD function, assuming the shipment is published successfully.

If the O_message from PROCESS_QUEUE_RECORD is NULL then, send NO_MSG in the status_code otherwise send the NEW_MSG in the status_code with the shipment number as business object Id. Also, send the message type as "asnoutcre".

Function Level Description – PUB_RETRY

```
PUB_RETRY (O_status_code    IN OUT   VARCHAR2,
           O_error_message   IN OUT   RTK_ERRORS.RTK_TEXT%TYPE,
           O_message_type    IN OUT   VARCHAR2,
           O_message         IN 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)
```

This procedure republishes the entity that failed to be published before. It is the same as GETNXT except that the record on SHIPMENT_PUB_INFO to be published must match the passed in sequence number contained in the ROUTING_INFO.

Function Level Description – PROCESS_QUEUE_RECORD (local)

```
PROCESS_QUEUE_RECORD (
      O_error_message IN OUT        RTK_ERRORS.RTK_TEXT%TYPE,
      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,
      I_shipment      IN            SHIPMENT.SHIPMENT%TYPE,
      I_seq_no        IN            SHIPMENT_PUB_INFO.SEQ_NO%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.

- The correct thread for the business transaction will be calculated and written. Call API_LIBRARY. GET_RIB_SETTINGS to get the number of threads used for the publisher. Using the number of threads, and the business object ID (for example, shipment number), calculate the thread value.
- Build the header and detail object by calling BUILD_HEADER_OBJECT.
- Delete the current record from the queue (i.e. shipment_pub_info table) by calling UPDATE_QUEUE_REC function.

Function Level Description – BUILD_HEADER_OBJECT (local)

```
BUILD_HEADER_OBJECT
      (O_error_message       IN OUT         RTK_ERRORS.RTK_TEXT%TYPE,
       O_rib_asnoutdesc_rec  IN OUT         "RIB_ASNOutDesc_REC",
       O_routing_info        IN OUT NOCOPY  RIB_ROUTINGINFO_TBL,
       I_shipment            IN         SHIPMENT_PUB_INFO.SHIPMENT%TYPE)
```

- Take all necessary data from the SHIPMENT table for the current shipment and put it into a "RIB_ASNOutDesc_REC" object. In addition, publish a schedule_number of NULL and auto_receive_ind of 'N' to the "RIB_ASNOutDesc_REC" object.
- The routing information has to be sent to RIB through RIB_ROUTINGINFO_REC. This routing info is for FROM location, TO location and source application (RMS) from which RIB receives the information. The routing location type for the TO location will be set to 'V' for the non stockholding company stores (i.e. virtual stores). Else, it will be set to 'S'. This is to ensure that shipment to a virtual store is not routed to SIM.
- If the destination location is Store then, set the asn_type as 'C' (Customer Store) and get the information about the store by calling STORE_ATTRIB_SQL.GET_INFO. Else, set the asn_type to 'T' (wh transfer) and get the information about WH by calling WH_ATTRIB_SQL.GET_WH_INFO function.
- Call the BUILD_DETAIL_OBJECTS to get the details of the current shipment record.
- The container_qty is a required field on the RIB object. So, RMS sends 1 instead of NULL in SHIPMENT.NO_BOXES if it is NULL.

Function Level Description – BUILD_DETAIL_OBJECTS (local)

```
BUILD_DETAIL_OBJECTS
      (O_error_message          IN OUT   RTK_ERRORS.RTK_TEXT%TYPE,
       O_rib_asnoutdistro_tbl   IN OUT   "RIB_ASNOutDistro_TBL",
       I_shipment_rec           IN       SHIPMENT%ROWTYPE)
```

The function is responsible for building detail level Oracle Objects. It builds as many detail Oracle Object as it can given the passed in message type and business object keys.

- Fetch the detail records of the shipment from SHIPSKU for the given shipment number.
- If the distro_type is 'T' then, get the transfer details by calling the TSF_ATTRIB_SQL.GET_TSFHEAD_INFO function. Else, get the corresponding allocation details from the alloc_detail table for the current distro_no and to_location.
- If the freight_code is 'E'xpedite then, set the expedite flag to 'Y' otherwise 'N'.
- When the transfer type is Customer Order "CO", the corresponding customer order number and fulfillment order number from the ORDCUST table will be published in the distro record.

- Assign the above details into "RIB_ASNOutItem_REC", "RIB_ASNOutCtn_REC" and "RIB_ASNOutDistro_REC" records.
- Because the container_qty and container_id are the mandatory fields, RMS will send "1" for container_qty and "0" for container_id instead of NULL.

Function Level Description – LOCK_THE_BLOCK (local)

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.

Function Level Description – HANDLE_ERRORS (local)

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 SHIPMENT_PUB_INFO record back to the RIB in the ROUTING_INFO. It sends back a status of 'H' - Hospital to the RIB as well. It then updates the status of the queue record to 'H', 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' - Error 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' to 'E'.

Function Level Description – UPDATE_QUEUE_REC (local)

UPDATE_QUEUE_REC is called from PROCESS_QUEUE_RECORD once a queue record is formed from SHIPMENT_PUB_INFO table. This will update the pub_status to 'P' so as not to pick-up the same record again.

Trigger Impact

Trigger name: EC_TABLE_SPT_AIR

Trigger file name: ec_table_spt_air.trg

Table: SHIPMENT_PUB_TEMP

A trigger on the SHIPMENT_PUB_TEMP table will capture the inserts.

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type **asnoutcre**.

Message XSD

Here is the filename that corresponds with the message type. Please consult the RIB documentation for this message type in order to get a detailed picture of the composition of the message.

Message Types	Message Type Description	XML Schema Definition (XSD)
asnoutcre	ASN Out Create Message	ASNOutDesc.xsd

Design 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.
- ASNOut messages published from RMS should NOT go back to RMS again.
- ASNOut messages published from RMS are intended for execution systems like SIM and RWMS. They are never routed to Order Management System (OMS). OMS is responsible for managing the order through its lifecycle from capture at the Online Order Capture (OOC) through fulfillment.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
SHIPMENT	Yes	No	No	No
SHIPSKU	Yes	No	No	No
SHIPMENT_PUB_INFO	Yes	Yes	Yes	No
ORDCUST	Yes	No	No	No
TSFHEAD	Yes	No	No	No
ALLOC_DETAIL	Yes	No	No	No

Banner Publication API

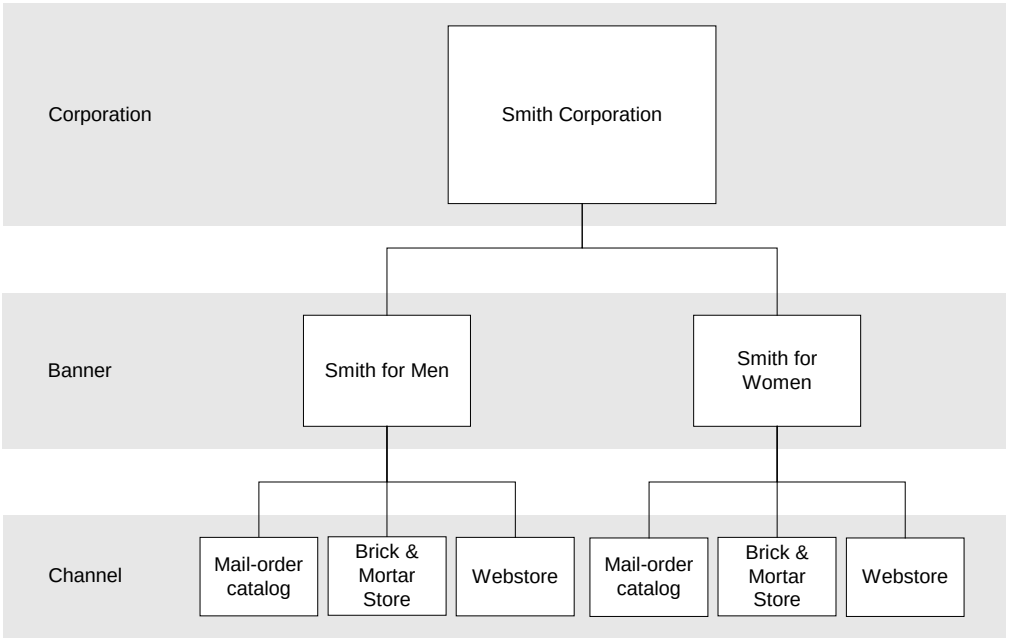
Functional Area

Foundation

Business Overview

RMS publishes messages about banners and channels to the Oracle Retail Integration Bus (RIB). A banner provides a means of grouping channels thereby allowing the customer to link all brick and mortar stores, catalogs, and web stores. The BANNER table holds a banner identifier and name. The CHANNELS table shows all channels and any associated banner identifiers.

The diagram Banners and Channels within a Corporation shows a sample of the structure of banners and channels within a corporation.



Banners and Channels within a Corporation

Banner/channel publication consists of a single flat message containing information from the tables BANNER and CHANNELS. One message is synchronously created and placed in the message queue each time a record is created, modified, or deleted. When a record is created or modified, the flat message contains several attributes of the banner/channel. When a record is deleted, the message contains the unique identifier of the banner/channel. Messages are retrieved from the message queue in the order they were created.

Package Impact

Create

1. **Prerequisites**: For channel creation, the associated banner must have been created.
2. **Activity Detail**: Once a banner/channel has been created, it is ready to be published. An initial publication message is made.
3. **Messages**: A “Banner Create” / “Channel Create” message is queued. This message is a flat message that contains a full snapshot of the attributes on the BANNER or CHANNEL table.

Modify

1. **Prerequisites**: banner/channel has been created.
2. **Activity Detail**: The user is allowed to change attributes of the banner/channel. These changes are of interest to other systems and so this activity results in the publication of a message.
3. **Messages**: Any modifications will cause a “banner modify” / channel modify” message to be queued. This message contains the same attributes as the “banner create” / “channel create” message.

Delete

1. **Prerequisites**: banner/channel has been created.
2. **Activity Detail**: Deleting a banner/channel removes it from the system. External systems are notified by a published message.
3. **Messages**: When a banner/channel is deleted, a "Banner Delete" / "Channel Delete" message, which is a flat notification message, is queued. The message contains the banner/channel identifier.

Package name: RMSMFM_banner

Spec file name: rmsmfm_banners.pls

Body file name: rmsmfm_bannerb.pls

Package Specification – Global Variables

None

Function Level Description – ADDTOQ

```
Procedure: ADDTOQ
           (O_status          OUT          VARCHAR2,
            O_text            OUT          VARCHAR2,
            I_message_type    IN           BANNER_MFQUEUE.MESSAGE_TYPE%TYPE,
            I_banner_id       IN           CHANNELS.BANNER_id%TYPE,
            I_channel_id      IN           CHANNELS.CHANNEL_ID%TYPE,
            I_message         IN           CLOB)
```

This procedure is called by the triggers EC_TABLE_BAN_AIUDR and EC_TABLE_CHN_AIUDR, and takes the message type, banner ID, channel ID (NULL if called from EC_TABLE_BAN_AIUDR) and the message itself. It inserts a row into the message family queue BANNER_MFQUEUE along with the passed in values and the next sequence number from the message family sequence, setting the status to unpublished. 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     CLOB,
               O_banner_id           OUT     NUMBER,
               O_channel_id          OUT     NUMBER)
```

This publicly exposed procedure is typically called by a RIB publication adaptor. Its parameters are well defined and arranged in a specific order. The message type is the RIB defined short message name. The message is the XML message. The family key consists of the banner ID, which will be populated for all message types, and the channel ID, which can be NULL.

The error text parameter contains application-generated information, such as the application's sequence number of the message that failed, and the Oracle or other error that occurred when the retrieval failed.

Function Level Description – GETNXT(local)

This procedure fetches the row from the message queue table that has the lowest sequence number. The message is retrieved, and then the row is removed from the queue.

Trigger Impact

Trigger exists on the banner and channels tables to capture inserts, updates, and deletes.

Trigger name: EC_TABLE_BAN_AIUDR.TRG

Trigger file name: ec_table_ban_aiudr.trg

Table: BANNER

This trigger captures inserts/updates/deletes to the BANNER table and writes data into the BANNER_MFQUEUE message queue. It calls BANNER_XML.BUILD_MESSAGE to create the XML message, and then calls RMSMFM_BANNER.ADDTOQ to insert this message into the message queue.

- **Inserts**: Sends banner info to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.BannerDesc and the original message.
- **Updates**: Sends banner info to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.BannerDesc and the original message
- **Deletes**: Sends banner info to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.BannerRef and the original message.

Trigger name: EC_TABLE_CHN_AIUDR.TRG

Trigger file name: ec_table_chn_aiudr.trg

Table: CHANNELS

This trigger captures inserts/updates/deletes to the CHANNELS table and writes data into the BANNER_MFQUEUE message queue. It calls CHANNEL_XML.BUILD_MESSAGE to create the XML message, and then calls RMSMFM_BANNER.ADDTOQ to insert this message into the message queue.

- **Inserts**: Sends channel info to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.ChannelDesc and the original message.
- **Updates**: Sends channel info to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.ChannelDesc and the original message.
- **Deletes**: Sends channel info to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.ChannelRef and the original message.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
BannerCre	Banner Create Message	BannerDesc.xsd
BannerMod	Banner Modify Message	BannerDesc.xsd

Message Types	Message Type Description	XML Schema Definition (XSD)
BannerDel	Banner Delete Message	BannerRef.xsd
ChannelsCre	Channels Create Message	ChannelDesc.xsd
ChannelsMod	Channels Modify Message	ChannelDesc.xsd
ChannelsDel	Channels Delete Message	ChannelRef.xsd

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
BANNER_MFQUEUE	Yes	Yes	No	Yes

Design Assumptions

One of the primary assumptions in the current approach is that ease of code will outweigh performance considerations. It is hoped that the 'trickle' nature of the flow of data will decrease the need to dwell on performance issues and instead allow developers to code in the easiest and most straight forward manner.

Customer Order Fulfillment Confirmation Publication API

Functional Area

Customer Order

Business Overview

When RMS is integrated with an external OMS, one of the supported deployment methods is interfacing customer order fulfillment requests into RMS through the RIB JMS. When RMS processes customer order requests, it will also publish a confirmation message containing the following information:

- Customer order number
- Fulfillment order number
- Confirm Type – 'C' (order fully created), 'P' (order partially created), or 'X' (order not created)
- Confirm number – PO or Transfer in RMS
- Item
- Reference Item
- Confirm quantity
- Confirm quantity UOM

Package Impact

Business Object ID

A customer order associated with an ordcust_no on ORDCUST is the business object to be published through this API.

Package name: RMSMFM_ORDCUST

Spec file name: rmsmfm_ordcusts.pls

Body file name: rmsmfm_ordcustb.pls

Package Specification – Global Variables

```
FAMILY        RIB_SETTINGS.FAMILY%TYPE := 'fulfilordcfm';
LP_cre_type   RIB_TYPE_SETTINGS.TYPE%TYPE := 'fulfilordcfmcre';
```

Function Level Description – ADDTOQ

```
ADDTOQ(O_error_message  IN OUT   RTK_ERRORS.RTK_TEXT%TYPE,
       I_message_type   IN       ORDCUST_PUB_INFO.MESSAGE_TYPE%TYPE,
       I_ordcust_no     IN       ORDCUST.ORDCUST_NO%TYPE)
```

- A trigger on the ORDCUST_PUB_TEMP table will call this function to add the customer order number to the ORDCUST_PUB_INFO table for publishing to the RIB. Only the create message type ('fulfilordcfmcre') is supported.
- Validate all the input parameters to this function against NULL. If any has NULL value then return from the function with the appropriate error message.
- Insert a record in the ORDCUST_PUB_INFO table. The published flag will be set to 'U'. The correct thread for the business transaction will be calculated and written. Call API_LIBRARY. GET_RIB_SETTINGS to get the number of threads used for the publisher. Using the number of threads, and the business object ID (for example, customer order number) calculate the thread value.

Function Level Description – GETNXT

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

Initialize LP_error_status to API_CODES.HOSPITAL at the beginning of GETNXT.

The RIB calls GETNXT to get messages. It performs a cursor loop on the unpublished records on the ORDCUST_PUB_INFO 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 (ORDCUST_PUB_INFO) for the current business object. The lock is obtained by calling the function LOCK_THE_BLOCK. If the record for the current business object is locked, the current message is skipped.
2. The information from the ORDCUST_PUB_INFO table is passed to PROCESS_QUEUE_RECORD. PROCESS_QUEUE_RECORD will build the RIB Object message to pass back to the RIB. If PROCESS_QUEUE_RECORD does not run successfully, GETNXT raises an exception.
3. If any exception is raised in GETNXT, including the exception raised by an unsuccessful call to PROCESS_QUEUE_RECORD, HANDLE_ERRORS is called.
4. Unconditionally exit from the loop after the successful processing of PROCESS_QUEUE_RECORD function, assuming the confirmation message is published successfully.

The loop will need to execute more than once if the record is locked on the queue table for the current business object.

Function Level Description – PUB_RETRY

```
PUB_RETRY(O_status_code     IN OUT   VARCHAR2,
          O_error_message   IN OUT   VARCHAR2,
          O_message_type    IN OUT   VARCHAR2,
          O_message         IN 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)
```

This procedure republishes the entity that failed to be published before. It is the same as GETNXT except that the record on ORDCUST_PUB_INFO to be published must match the passed in sequence number contained in the ROUTING_INFO.

Function Level Description – PROCESS_QUEUE_RECORD (local)

```
PROCESS_QUEUE_RECORD(
       O_error_message  IN OUT         RTK_ERRORS.RTK_TEXT%TYPE,
             O_message        IN OUT NOCOPY  RIB_OBJECT,
             O_routing_info   IN OUT NOCOPY  RIB_ROUTINGINFO_TBL,
             I_ordcust_no     IN    ORDCUST_PUB_INFO.ORDCUST_NO%TYPE,
             I_seq_no         IN            ORDCUST_PUB_INFO.SEQ_NO%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.

- The correct thread for the business transaction will be calculated and written. Call API_LIBRARY. GET_RIB_SETTINGS to get the number of threads used for the publisher. Using the number of threads, and the business object ID (for example, customer order number), calculate the thread value.
- Build the header and detail object by calling BUILD_MSG_OBJECT.
- Update the pub_status to 'P' for the current record in the ORDCUST_PUB_INFO table.
- Delete the current record in the ORDCUST_PUB_TEMP table.
- Set LP_error_status to API_CODES.UNHANDLED_ERROR before any DML statements.

No routing information will be included since all published messages will go to OMS and no other applications.

Function Level Description – BUILD_MSG_OBJECT (local)

Take all necessary data from the ORDCUST, ORDCUST_DETAIL, ORDHEAD, ORDLOC, TSFHEAD, and TSFDETAIL tables and put into a "RIB_FulfilOrdCfmDesc_REC" object.

Function Level Description – LOCK_THE_BLOCK (local)

This function locks the record 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.

Function Level Description – HANDLE_ERRORS (local)

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 ORDCUST_PUB_INFO record back to the RIB in the ROUTING_INFO. It sends back a status of 'H' - Hospital to the RIB as well. It then updates the status of the queue record to 'H', 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' - Error 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' to 'E'.

Trigger Impact

Trigger name: EC_TABLE_ORP_AIR

Trigger file name: ec_table_orp_air.trg

Table: ORDCUST_PUB_TEMP

The trigger ORDCUST_PUB_TEMP table will capture inserts and send the appropriate column values to the ADDTOQ procedure in the MFM with message type RMSMFM_ORDCUST.LP_cre_type.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
fulfilordcfmcre	Customer Order Fulfillment Confirmation Create Message	FulfilOrdCfmDesc.xsd

Design Assumptions

- RMS will only publish confirmation 'create' messages associated to a PO or transfer.
- 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.
- OMS is the only subscriber of this message family. Since all published customer order fulfillment confirmation messages will be routed to OMS, no routing info is needed.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ORDCUST_PUB_INFO	Yes	Yes	Yes	No
ORDCUST_PUB_TEMP	Yes	No	No	Yes
ORDCUST	Yes	No	No	No
ORDCUST_DETAIL	Yes	No	No	No
ORDHEAD	Yes	No	No	No
ORDLOC	Yes	No	No	No
TSFHEAD	Yes	No	No	No
TSFDETAIL	Yes	No	No	No

Delivery Slot Publication API

Functional Area

Replenishment

Business Overview

RMS provides retailers the option of creating store orders for items with multiple delivery instructions per day for the same item. RMS provides this multiple deliveries per day support by generating multiple purchase orders and/or transfers based on need day and delivery slot.

Since the replenishment batch can be run during the day time, it is necessary to lock the important transaction tables. The following tables are locked for the intraday replenishment:

- TSF_DETAIL
- ITEM_LOC_SOH
- ORD_IMV_MGMT
- CONTRACT_DETAIL
- CONTRACT_HEAD
- DEAL_HEAD
- ALLOC_CHRG
- ALLOC_HEADER
- ALLOC_DETAIL
- ORDLOC
- ORDLOC_REV
- ORDLOC_WKSHT
- ORDLOC_EXP
- ORDCUST
- ORDHEAD_REV
- ORDSKU
- REQ_DOC
- TIMELINE
- ORDLC
- DEAL_ITEMLOC_DIV_GRP
- DEAL_ITEMLOC_DCS
- DEAL_ITEMLOC_ITEM
- DEAL_ITEMLOC_PARENT_DIFF
- DEAL_THRESHOLD
- DEAL_DETAIL
- DEAL_QUEUE
- DEAL_CALC_QUEUE
- REV_ORDERS

Delivery slot ID publication consists of a single flat message containing the delivery slot details from the table DELIVERY_SLOT. One message will be synchronously created and placed in the message queue each time a delivery_slot_id is created, updated or deleted from delivery_slot. When a delivery_slot_id is created or deleted, the flat message will contain 3 attributes i.e delivery_slot_id, deliver_slot_desc and delivery_slot_sequence. Messages are retrieved from the message queue in the order they were created.

Package Impact

Create Delivery_Slot

1. **Prerequisites**: Delivery_slot does not already exist.
2. **Activity Detail**: Any insert to the DELIVERY_SLOT table inserts a 'dlvysltcre' message_type record on the DELIVERY_SLOT_MFQUEUE table.

Update Delivery_Slot

1. **Prerequisites**: Delivery_slot does already exist.
2. **Activity Detail**: Any update to the DELIVERY_SLOT table inserts a 'dlvysltmod' message_type record on the DELIVERY_SLOT_MFQUEUE table.

Delete Delivery_slot

1. **Prerequisites**: Delivery_slot already exist.
2. **Activity Detail**: Deleting a delivery_slot_id removes the record from the delivery_slot table and inserts a 'dlvysltdel' row to the DELIVERY_SLOT_MFQUEUE table.

Package name: RMSMFM_DLVYSLT

Spec file name: rmsmfm_dlvyslts.pls

Body file name: rmsmfm_dlvysltb.pls

Package Specification – Global Variables

FAMILY CONSTANT RIB_SETTINGS.FAMILY%TYPE: = 'dlvyslt';

SLT_ADD CONSTANT VARCHAR2 (15): = 'dlvysltcre';

SLT_UPD CONSTANT VARCHAR2 (15) := 'dlvysltmod';

SLT_DEL CONSTANT VARCHAR2 (15): = 'dlvysltdel';

Function Level Description – ADDTOQ

Function:

```
ADDTOQ
(O_status              OUT   VARCHAR2,
 O_text                OUT   VARCHAR2,
 I_message_type        IN    DELIVERY_SLOT_MFQUEUE.MESSAGE_TYPE%TYPE,
 I_delivery_slot_id    IN    DELIVERY_SLOT_MFQUEUE.DELIVERY_SLOT_ID%TYPE,
 I_delivery_slot_desc  IN    DELIVERY_SLOT_MFQUEUE.DELIVERY_SLOT_DESC%TYPE,
 I_delivery_sequence   IN    DELIVERY_SLOT_MFQUEUE.DELIVERY_SLOT_SEQUENCE%TYPE);
```

An event capture trigger calls this procedure with the message type for synchronously captured messages. It inserts a row into the message family queue along with the passed in values, the next sequence number from the message family sequence, and a status of unpublished. Due to the very small data volume of delivery slots, no multi-threading is supported for this publishing. Therefore, the thread_no is always set to 1. It returns the standard publishing API success or failure codes.

Function Level Description – GETNXT

```
Procedure: GETNXT
           (O_status_code     IN OUT   VARCHAR2,
           O_error_msg        IN OUT   RTK_ERRORS.RTK_TEXT%TYPE,
           O_message_type     IN OUT   DELIVERY_SLOT_MFQUEUE.MESSAGE_TYPE%TYPE,
           O_message          IN OUT   RIB_OBJECT,
           O_bus_obj_id       IN OUT   RIB_BUSOBJID_TBL,
           O_routing_info     IN OUT   RIB_ROUTINGINFO_TBL,
           I_num_threads      IN       NUMBER DEFAULT 1
           I_thread_val       IN       NUMBER DEFAULT 1);
```

This procedure is publically available and is typically called by a RIB publication adaptor. Its parameters are well defined and arranged in a specific order. The message type is the RIB defined short message name ('dlvyslt') and the message is a RIB object ("RIB_DeliverySlotDesc_REC" for a create and update message, "RIB_DeliverySlotRef_REC" for a delete message).

The error text parameter contains application-generated information, such as the application's sequence number of the message that failed, and the Oracle or other error that occurred when the retrieval failed.

This program loops through each message on the DELIVERY_SLOT_MFQUEUE table, and calls PROCESS_QUEUE_RECORD. When no messages are found, the program exits returning the 'N'o message found API code.

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 DELIVERY_SLOT_MFQUEUE table. The record on DELIVERY_SLOT_MFQUEUE must match the sequence number passed in routing info data structure.

Function Level Description – PROCESS_QUEUE_DLVY_SLT (local)

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 **DELIVERY_SLOT_MFQUEUE** is a create or update message then

- Build and pass the RIB_DeliverySlotDesc_REC object
- Delete the record from the delivery_slot_mfqueue table.

If the record from **DELIVERY_SLOT_MFQUEUE** table is a delete then

- Build and pass the RIB_DeliverySlotRef_REC object.
- Delete the record from the delivery_slot_mfqueue table.

Trigger Impact

Create a trigger on Delivery_Slot table to capture inserts and deletes.

Trigger name: EC_TABLE_DLVY_AIUDR.TRG

Trigger file name: ec_table_dlvy_aiudr.trg

Table: Delivery_Slot

- **Inserts**: Send the I_delivery_slot_id, I_delivery_slot_desc, I_delivery_sequence and a message type of 'dlvysltcre' to the ADDTOQ procedure.
- **Updates**: Send the I_delivery_slot_id, I_delivery_slot_desc, I_delivery_sequence and a message type of 'dlvysltmod' to the ADDTOQ procedure.
- **Deletes**: **Send** the I_delivery_slot_id, I_delivery_slot_desc, I_delivery_sequence and a message type of 'dlvysltdel' to the ADDTOQ procedure-.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
Dlvysltcre	Delivery slot Create Message	DeliverySlotDesc.xsd
Dlvysltdel	Delivery slot delete Message	DeliverySlotRef.xsd

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
DELIVERY_SLOT_MFQUEUE	YES	YES	Yes	YES

Design Assumptions

- It is not possible for the trigger to know the status of anything modified by GETNXT. If a 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 (at most 2-3 seconds). This also has to occur at the same time GETNXT is processing the current business object.
- Delay all DML statements to as late a time as possible. Once DML statements have taken place, any error becomes a fatal error rather than a hospital error.

Differentiator Groups Publication API

Functional Area

Foundation

Business Overview

Differentiator (Diff) Group publication consists of a single flat message containing diff group attributes from the tables DIFF_GROUP_HEAD and DIFF_GROUP_DETAIL. A message is synchronously created and placed in the message queue each time a diff group (DIFF_GROUP_HEAD) is created, modified, or deleted or when a diff (DIFF_GROUP_DETAIL) is created, modified, or deleted from a diff group. When a diff group (DIFF_GROUP_HEAD) is created or modified, the flat message contains numerous attributes of the group. When a diff group is deleted, the message contains the both unique identifier of the group, and the diff_group_id. When a diff (diff_group_detail) is created or modified, the flat message contains numerous attributes of the diff. When a diff is deleted, the message contains the unique identifier of the diff group and the diff, diff_group_id and diff_id. A Message is retrieved from the message queue in the order they were created.

Package Impact

Create Diff Group

1. **Prerequisites**: Diff Group does not already exist.
2. **Activity Detail**: Any change to the DIFF_GROUP_HEAD table inserts a DiffGrpHdrCre message_type record on the DIFFGRP_MFQUEUE table.
3. **Messages**: The DiffGrpHdrDesc message is created. It is a flat, synchronous message containing a full snapshot of the diff group at the time the message is published.

Modify Diff Group

1. **Prerequisites**: Diff Group exists.
2. **Activity Detail**: Any change to the DIFF_GROUP_HEAD table inserts a DiffGrpHdrMod message_type record on the DIFFGRP_MFQUEUE table.
3. **Messages**: The DiffGrpHdrDesc message is created. It is a flat, synchronous message containing a full snapshot of the diff group at the time the message is published.

Create Diff Group Detail

1. **Prerequisites**: A Diff Group already exists, and the diff ID exists on diff_ids, but the diff ID does not exist within the diff group.
2. **Activity Detail**: Any Differentiators added to a diff group inserts a record to the DIFF_GROUP_HEAD table. A DiffGrpDtlCre message type record is also inserted on the DIFFGRP_MFQUEUE table. A foreign key to the DIFF_GROUP_HEAD table checks the existence of the diff group the value is created to supplement.
3. **Messages**: DiffGrpDtlDesc message type is created. It is a hierarchical, synchronous message containing a snapshot of the DIFF_GROUP_DETAIL table at the time the message is published.

Modify Diff Group Detail

1. **Prerequisites**: Diff Group and the Diff ID within the diff group (DIFF_GROUP_DETAIL record) exist.
2. **Activity Detail**: Any change to the diffs within a diff group modifies a record to the DIFF_GROUP_HEAD table. A DiffGrpDtlMod message type record is also inserted on the DIFFGRP_MFQUEUE table. A foreign key to the DIFF_GROUP_HEAD table checks the existence of the diff group the value is created to supplement.
3. **Messages** DiffGrpDtlDesc message is created. It is a flat, synchronous message containing a snapshot of the DIFF_GROUP_DETAIL table at the time the message is published.

Delete Diff Group Detail

1. **Prerequisites**: Diff Group and the Diff ID within the diff group (DIFF_GROUP_DETAIL record) exist.
2. **Activity Detail**: Deleting a diff from a Diff Group removes it from the DIFF_GROUP_DETAIL table and inserts a DiffGrpDtlDel row to the DIFFGRP_MFQUEUE table.
3. **Message**: A DiffGrpDtlRef message is created. It is a flat, synchronous message containing the primary key with which the external systems can remove it from their systems.

Delete Diff Group

1. **Prerequisites**: Diff Group exists and a diff ID within the diff group (DIFF_GROUP_DETAIL record) may or may not exist.
2. **Activity Detail**: Deleting a Diff Group removes it from the DIFF_GROUP_HEAD table and inserts a DiffGrpDel row to the DIFFGRP_MFQUEUE table. Because the Diff Group Maintenance form in RMS automatically removes any child records on the DIFF_GROUP_DETAIL table when the diff group is removed, there will be a row inserted to the DIFFGRP_MFQUEUE table for each DIFF_GROUP_DETAIL record associated with the deleted diff group as well. These will receive the lower sequence numbers so that these will be acted upon first in the message queue. They will look like the DELETE DIFF_GROUP_DETAIL message detailed in the section above.
3. **Message**: A DiffGrpRef message is created for the diff group only. It is a flat, synchronous message containing the primary key with which the external systems can remove it from their systems.

Package name: RMSMFM_DIFFGRP

Spec file name: rmsmfm_diffgrps.pls

Body file name: rmsmfm_diffgrpb.pls

Function Level Description – ADDTOQ

```
Function: ADDTOQ
           (O_status          OUT           VARCHAR2,
            O_text            OUT           VARCHAR2,
            I_message_type    IN            DIFFGRP_MFQUEUE.MESSAGE_TYPE%TYPE
            I_diff_group_id   IN            DIFFGRP_MFQUEUE.DIFF_GROUP_ID%TYPE,
            I_diff_id         IN            DIFFGRP_MFQUEUE.DIFF_ID%TYPE,
            I_message         IN            CLOB);
```

This procedure is called by an event capture trigger, and takes the message type, family key values and, for synchronously captured messages, the message itself. It inserts a row into the message family queue along with the passed in values and the next sequence number from the message family sequence, setting the status to unpublished, or skips in the case of consolidation messages. It returns error codes and strings according to the standards of the application in which it is being implemented.

Function Level Description – GETNXT

```
Procedure: GETNXT
           (O_status_code     OUT     VARCHAR2,
            O_error_msg       OUT     VARCHAR2,
            O_message_type    OUT     VARCHAR2,
            Message           OUT     CLOB,
            O_diff_group_id   OUT     DIFFGRP_MFQUEUE.DIFF_GROUP_ID%TYPE,
            O_diff_id         OUT     DIFFGRP_MFQUEUE.DIFF_ID%TYPE);
```

This publicly exposed procedure is typically called by a RIB publication adaptor. Its parameters are well defined and arranged in a specific order. The message type is the RIB defined short message name, the message is the xml message, and the family key(s) are the key for the message as pertains to the family, not all of which will necessarily be populated for all message types. Status code is the API_CODES which denotes the success or failure of processing the message.

The error text parameter contains application-generated information, such as the application's sequence number of the message that failed, and the Oracle or other error that occurred when the retrieval failed. The facility ID is only included in messages coming from RWMS.

Trigger Impact

A trigger exists on the DIFF_GROUP_HEAD and DIFF_GROUP_DETAIL table to capture inserts, updates, and deletes.

Trigger name: EC_TABLE_DGH_AIUDR.TRG

Trigger file name: ec_table_dgh_aiudr.trg

Table: DIFF_GROUP_HEAD

- **Inserts**: Send the I_diff_group_id to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.
- **Updates**: Send the I_diff_group_id to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY. Any change to the DIFF_GROUP_HEAD table inserts a DiffGrpHdrCre message_type record on the DIFFGRP_MFQUEUE table.
- **Deletes**: Send the I_diff_group_id to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.

Trigger name: EC_TABLE_DGD_AIUDR.TRG

Trigger file name: ec_table_dgd_aiudr.trg

Table: DIFF_GROUP_DETAIL

- **Inserts:** Send the I_diff_group_id, I_diff_id to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.
- **Updates** :
    - Send the I_diff_group_id, I_diff_id to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.
    - Any Differentiators added to a diff group inserts a record to the DIFF_GROUP_HEAD table. A DiffGrpDtlCre message type record is also inserted on the DIFFGRP_MFQUEUE table. A foreign key to the DIFF_GROUP_HEAD table checks the existence of the diff group the value is created to supplement.
- **Deletes**: Send the I_diff_group_id, I_diff_id to the ADDTOQ procedure in the MFM with the message type RMSMFM_FAMILY.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
DiffGrpHdrCre	Differentiator Header Create Message	DiffGrpHdrDesc.xsd
DiffGrpHdrMod	Differentiator Header Modify Message	DiffGrpHdrDesc.xsd
DiffGrpHdrDel	Differentiator Header Delete Message	DiffGrpHdrRef.xsd
DiffGrpDtlCre	Differentiator Detail Create Message	DiffGrpDtlDesc.xsd
DiffGrpDtlMod	Differentiator Detail Modify Message	DiffGrpDtlDesc.xsd
DiffGrpDtlDel	Differentiator Detail Delete Message	DiffGrpDtlRef.xsd

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
DIFFGRP_MFQUEUE	YES	YES	NO	YES

Design Assumptions

- It is not possible for a detail trigger to accurately know the status of a header table.
- 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 (at most 2-3 seconds). It is also assumed that this will occur rarely because 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.
- Delay all DML statements to as late a time as possible. Once DML statements have taken place, any error becomes a fatal error rather than a hospital error.

Differentiator ID Publication API

Functional Area

Foundation

Business Overview

RMS publishes messages for differentiator (diff) identifiers (diff IDs), and diff groups.

When diff are created in RMS and need to be sent to other systems, they are sent out via diff ID publication. When the external system receives information about an item that includes the new diff ID, that system understands what the diff ID refers to.

Diff message processes

Diff message publication processes begin whenever a trigger 'fires' on one of the diff tables. When that occurs, the trigger extracts the affected row on the table and publishes the data to the corresponding message family queue staging table. A total of nine messages can be published; however, they group into these three categories:

- Group Header
- Group Details
- Diff IDs

Diff ID publication consists of a single flat message containing diffattributes from the table DIFF_IDS. One message will be synchronously created and placed in the message queue each time a diff (diff_ids) is created, modified, or deleted. When a diff (diff_ids) is created or modified, the flat message will contain numerous attributes of the diff. When a diff is deleted, the message will simply contain the unique identifier of the diff, the diff_id. Messages are retrieved from the message queue in the order they were created.

Package Impact

Create Diff Id:

1. **Prerequisites**: Diff ID does not already exist.
2. **Activity Detail**: Any change to the DIFF_IDS table inserts a DiffCre message_type record on the DIFFID_MFQUEUE table.
3. **Messages**: The DiffDesc message is created. It is a flat, synchronous message containing a full snapshot of the diff ID at the time the message is published.

Modify Diff Id

1. **Prerequisites**: Diff ID exists.
2. **Activity Detail**: Any change to the DIFF_IDS table inserts a DiffMod message_type record on the DIFFID_MFQUEUE table.
3. **Messages**: The DiffDesc message is created. It is a flat, synchronous message containing a full snapshot of the diff ID at the time the message is published.

Delete Diff Id

1. **Prerequisites**: Diff ID exists.
2. **Activity Detail**: Deleting a Diff ID removes it from the DIFF_IDS table and inserts a DiffDel row to the DIFFID_MFQUEUE table.
3. **Message**: A DiffRef message is created. It is a flat, synchronous message containing the primary key with which the external systems can remove it from their systems.

Package name: RMSMFM_DIFFID

Spec file name: rmsmfm_diffids.pls

Body file name: rmsmfm_diffidb.pls

Package Specification – Global Variables

None

Function Level Description – ADDTOQ

```
Function: ADDTOQ(O_status          OUT      VARCHAR2,
                 O_text            OUT      VARCHAR2,
                 I_message_type    IN       DIFFID_MFQUEUE.MESSAGE_TYPE%TYPE,
                 I_diff_id         IN       DIFFID_MFQUEUE.DIFF_ID%TYPE,
                 I_message         IN       CLOB)
```

This procedure called by EC_TABLE_DID_AIUDR , takes the message type, diff ID, and the message itself. It inserts a row into the message family queue DIFFID_MFQUEUE along with the passed in values and the next sequence number from the message family sequence, setting the status to unpublished. 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     CLOB,
                  O_diff_id          OUT     DIFFGRP_MFQUEUE.DIFF_ID%TYPE)
```

This publicly exposed procedure is typically called by a RIB publication adaptor. Its parameters are well defined and arranged in a specific order. The message type is the RIB defined short message name, the message is the xml message, and the family key(s) are the key for the message as pertains to the family, not all of which will necessarily be populated for all message types.

The error text parameter contains application-generated information, such as the application's sequence number of the message that failed, and the Oracle or other error that occurred when the retrieval failed.

Function Level Description – GETNXT(local)

This procedure fetches the row from the message queue table that has the lowest sequence number. The message is retrieved, and then the row is removed from the queue.

Trigger Impact

A trigger exists on the DIFF_IDS and DIFFID_MFQUEUE tables to capture Inserts, Updates, and Deletes.

Trigger name: EC_TABLE_DID_AIUDR.TRG

Trigger file name: ec_table_did_aiudr.trg

Table: DIFF_IDs

DIFFID_XML. BUILD_MESSAGE (O_status, O_text, O_message, I_record, I_action_type) – This function is called by the trigger EC_TABLE_DID_AIUDR on insert, update and delete of the DIFF_IDS table. This function gathers all the data necessary to build the message that needs to be sent to the Oracle Retail Integration Bus. It determines the proper message to build based on the action_type that is sent in the trigger. It builds DiffRef xml messages for delete statements or DiffDesc xml messages for updates or inserts.

- **Inserts**: Sets action_type to 'A'dd and message_type to 'DiffCre'.
- **Updates**: Sets action_type to 'M'odify and message_type to 'DiffMod'.
- **Deletes**: Sets action_type to 'D'elete and message_type to 'DiffDel'.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
DiffCre	Diffid Create Message	DiffDesc.xsd
DiffMod	Diffid Modify Message	DiffDesc.xsd
DiffDel	Diffid Delete Message	DiffRef.xsd

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
DIFFID_MFQUEUE	Yes	Yes	No	Yes

Design Assumptions

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

Item Publication API

Functional Area

Foundation

Business Overview

RMS publishes messages about items to the Oracle Retail Integration Bus (RIB). In situations where a retailer creates a new item in RMS, the message that ultimately is published to the RIB contains a hierarchical structure of the item itself along with all components that are associated with that item. Items and item components make up what is called the Items message family.

After the item creation message has been published to the RIB for use by external applications, any modifications to the basic item or its components cause the publication of individual messages specific to that component. Deletion of an item and component records has similar effects on the message modification process, with the exception that the delete message holds only the key(s) for the record.

Deposit items

A deposit item is a product that has a portion which is returnable to the supplier and sold to the customer, with a deposit taken for the returnable portion. Because the contents portion of the item and the container portion of the item have to be managed in separate financial accounts (as the container item would be posted to a liabilities account) with different attributes, the retailer must set up two separate items. All returns of used deposit items (the returned item) are managed as a separate product, to track these products separately and as a generic item not linked to the actual deposit item (for example, bottles being washed and having no label).

The retailer can never put a container item on a transfer. Instead, the container item is added to returns to vendors (RTVs) automatically when the retailer adds the associated content item.

Deposit item attributes in RMS enable contents, container and crate items to be distinguished from one another. Additionally, it is possible to link a contents item to a container item for the purposes of inventory management.

In addition to contents and container items, many deposit items are delivered in plastic crates, which are also given to the customer on a deposit basis. These crates are sold to a customer as an additional separate product. Individual crates are not linked with contents or container items. Crates are specified in the system with a deposit item attribute.

From a receiving perspective, only the content item can be received. The receipt of a PO shows the container item but the receipt of a transfer does not. Similar to RTV functionality, online purchase order functionality automatically adds the container. The system automatically replicates all transactions for the container item in the stock ledger. In sum, for POs and RTVs, the container item is included; for transfers, no replication occurs.

Catch-Weight Items

Retailers can order and manage products for the following types of catch-weight item:

- **Type 1**: Purchase in fixed weight simple packs: sell by variable weight (for example, bananas).
- **Type 2**: Purchase in variable weight simple packs: sell by variable weight (for example, ham on the bone sold on a delicatessen counter).
- **Type 3**: Purchase in fixed weight simple packs containing a fixed number of eaches: sell by variable weight eaches (for example, pre-packaged cheese).
- **Type 4**: Purchase in variable weight simple packs containing a fixed number of eaches: sell by variable weight eaches (for example, pre-packaged sirloin steak).

> **Note:** Oracle Retail suggests that catch-weight item cases be managed through the standard simple pack functionality.

In order for catch-weight items to be managed in RMS, the following item attributes are available:

- **Cost UOM**: All items in RMS will be able to have the cost of the item managed in a separate unit of measure (UOM) from the standard UOM. Where this is in a different UOM class from the standard UOM, case dimensions must be set up.
- **Catch-weight item pack details**: Tolerance values and average case weights are stored for catch-weight item cases to allow the retailer to report on the sizes of cases received from suppliers.
- Maximum catch-weight tolerance threshold.
- Minimum catch-weight tolerance threshold.

Retailers can set up the following properties for a catch-weight item:

- Order type
- Sale type

Retailers can also specify the following, at the item-supplier-country level:

- Cost unit of measure (CUOM).

Receiving and inventory movement impact on catch-weight items

Inventory transaction messages include purchase order receiving, stock order receiving, returns to vendor, direct store delivery receiving, inventory adjustments and bill of lading. These messages include attributes that represent, for catch-weight items, the actual weight of goods involved in a transaction. These attributes are weight and weight UOM.

When RMS subscribes to inventory transaction messages containing such weight data, the transaction weight will be used for two purposes:

- To update weighted average cost (WAC) using the weight rather than the number of units and to update the average weight value of simple packs

> **Note:** The WAC calculation does not apply to return to vendors (RTVs).

Item Transformation

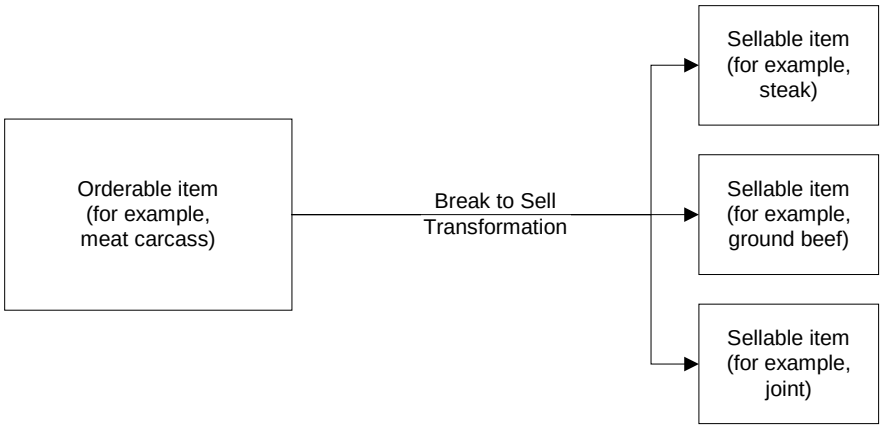
Item transformation allows retailers to manage items where the actual transformation of a product cannot be adequately recorded due to in-store processes.

With product transformation, new 'transform' items are set up as either sellable only or orderable only.

- **Sellable only items**: A sellable only item has no inventory in the system, so inventory records cannot be viewed from the item maintenance screens. Sellable only items do not hold any supplier links and therefore have no cost prices associated with them.
- **Orderable only items**: Orderable only items hold inventory, but are not sellable at the POS system. Therefore, no information is sent to the POS system for these items, and no unit retail prices by zone are held for these items.

To hold the relationship between the orderable items and the sellable items, RMS stores the transformation details. These details are used to process sales and inventory transactions for the items.

The following diagram shows how item transformation works:



Item transformation

Item and Item Component

The item message family is a logical grouping for all item data published to the RIB. The components of item messages and their base tables in RMS are:

- Item from the ITEM_MASTER table
- Item-supplier from ITEM_SUPPLIER
- Item-supplier-country from ITEM_SUPP_COUNTRY
- Item-supplier-country-dimension from ITEM_SUPP_COUNTRY_DIM (DIM is the each, inner, pallet, and case dimension for the item, as specified)
- Item-image from ITEM_IMAGE
- Item-UDA identifier-UDA value from UDA_ITEM_LOV (UDA is a user-defined attribute and LOV is list of values)
- Item-UDA identifier from UDA_ITEM_DATE (for the item and UDA date)
- Item-UDA identifier from UDA_ITEM_FF (for UDA, free-format data beyond the values for LOV and date)
- Item-pack components (Bill of Material [BOM]) from PACKITEM_BREAKOUT
- Item UPC reference from ITEM_MASTER.ITEM_NUMBER_TYPE (values held as code type 'UPCT' on code_head and code_detail tables)

- Item ticket from ITEM_TICKET
- Item relationship details from RELATED_ITEM_HEAD
- Related Items details from RELATED_ITEM_DETAIL

New Item Message Processes

The creation of a new item in RMS begins with an item in a worksheet status on the ITEM_MASTER table. At the time an item is created, other relationships are being defined as well, including the item, supplier, and country relationships, user-defined attributes (UDAs), related items and others. These item relationship processes in effect become components of a new item message published to the RIB. This section describes the item creation message process and includes the basic item message itself along with the other component relationship messages that become part of the larger item message.

Basic Item Message

As described in the preceding section, item messages can originate in a number of RMS tables. Each of these tables holds a trigger, which fires each time an insert, update, or delete occurs on the table. The new item record itself is displayed on the ITEM_MASTER table. The trigger on this table creates a new message (in this case, a message of the type ItemHdrCre), then calls the message family manager RMSMFM_ITEMS and its ADDTOQ public procedure. ADDTOQ populates the message to the ITEM_MFQUEUE staging table by inserting the following:

- Appropriate value into the message_type column.
- Message itself to the message column. Messages are of the data type CLOB (character large object).

New Item Message Publication

The publication of a new item and its components to the RIB is done using a hierarchical message. Here is how the process works:

1. A new item is held on ITEM_MASTER in a status of W (Worksheet) until it is approved.
2. On the ITEM_MFQUEUE staging table, a Worksheet status item is displayed in the message_type column as a value of ItemCre.
3. As the item continues to be built on ITEM_MASTER, an ItemHdrMod value is inserted into the queue's message_type column.
4. After the item is approved (ITEM_MASTER's status column value of A [Approved], the trigger causes the insertion of a value of Y (Yes) in the approve_ind column on the queue table.
5. A message with a top-level XML tag of ItemDesc is created that serves as a message wrapper.

At the same time, a sub-message with an XML tag of ItemHdrDesc is also created. This subordinate tag holds a subset of data about the item, most of which is derived from the ITEM_MASTER table.

Subordinate Data and XML Tags

While a new item is being created, item components are also being created. Described earlier in this overview, these component item messages pertain to the item-supplier, item-supplier-country, UDAs, and so on. For example, a new item-supplier record created on ITEM_SUPPLIER causes the trigger on this table to add an ItemSupCre value to the message_type column of the ITEM_MFQUEUE staging table. When the item is approved, a message with an XML tag of ItemSupDesc is added underneath the ItemDesc tag.

Similar processes occur with the other item components. Each component has its own Desc XML tag, for example: ItemSupCtyDesc, ISCDimDesc.

Modify and Delete Messages

Updates and deletions of item data can be included in a larger ItemDesc (item creation) message. If not part of a larger hierarchical message, they are published individually as a flat, non-hierarchical message. Update and delete messages are much smaller than the large hierarchy in a newly created item message (ItemDesc).

Modify Messages

If an existing item record changes on the ITEM_MASTER table, for example, the trigger fires to create an ItemHdrMod message and message type on the queue table. In addition, an ItemHdrDesc message is created. If no ItemCre value already exists in the queue, the ItemHdrDesc message is published to the RIB.

Similarly, item components like item-supplier that are modified, result in an ItemSupMod message type inserted on the queue. If an ItemCre and an ItemSupCre already exist, the ItemSupMod is published as part of the larger ItemDesc message. Otherwise, the ItemSupMod is published as an ItemSupDesc message.

Delete messages

Delete messages are published in the same way that modify messages are. For example, if an item-supplier-country relationship is deleted from RMS' ITEM_SUP_COUNTRY table, the dependent record on ITEM_SUPP_COUNTRY_DIM is also deleted.

1. An ItemSupCtyDel message type is displayed on the item queue table.
2. If the queue already holds an ItemCre or ItemSupCtyCre message, any ItemSupCtyCre and ItemSupCtyMod messages are deleted.

   Otherwise, ItemSupCtyDel is published by itself as an ItemSupCtyRef message to the RIB.

Design Overview

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

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

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

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

```
Family Header
item_master  -  transaction level items only
descriptions for the code values
names for department, class and subclass
diff types
base retail price
Item Family Child Tables
item_supplier
item_supp_country
item_supp_country_dim
    descriptions for the code values
item_master  -  reference items
    item, item_number_type, item_parent, primary_ref_ind, format_id, prefix
packitem_breakout
    pack_no, item, packitem_qty
item_image
item_ticket
uda_item_ff
uda_item_lov
uda_item_date
related_item_head
related_item_detail
```

Business Object Records

Create the following business objects to assist the publishing process:

1. Create a type for a table of rowids.

   ```
   TYPE ROWID_TBL is TABLE OF ROWID;
   ```

2. Create a record of ROWID_TBL types for keeping track of rowids to update and delete. There should be a ROWID_TBL for ITEM_MFQUEUE deletion, ITEM_MFQUEUE updating, ITEM_PUB_INFO deletion, and ITEMLOC_MFQUEUE deletion.

   ```
       TYPE ITEM_ROWID_REC is RECORD
   (queue_rowid_tbl       ROWID_TBL,
    pub_info_rowid_tbl    ROWID_TBL,
    queue_upd_rowid_tbl   ROWID_TBL,
    itemloc_rowid_tbl     ROWID_TBL
   );
   ```

3. Create a record to assist in publishing the ItemBOM node. This record type was originally in ITEMBOM_XML, but since ITEMBOM_XML is being removed, it is being moved to RMSMFM_ITEMS.

```
TYPE bom_rectype IS RECORD
(pack_no                    VARCHAR2(25),
 seq_no                     NUMBER(4),
 item                       VARCHAR2(25),
 item_parent                VARCHAR2(25),
 pack_tmpl_id               NUMBER(8),
 comp_pack_no               VARCHAR2(25),
 item_qty                   NUMBER(12,4),
 item_parent_pt_qty         NUMBER(12,4),
 comp_pack_qty              NUMBER(12,4),
 pack_item_qty              NUMBER(12,4));

TYPE bom_tabtype is TABLE of bom_rectype
INDEX BY BINARY_INTEGER;
```

Package Impact

Business Object ID

The business object ID for item publisher is item, which uniquely identifies an item for publishing.

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.

Item type X, item A, message type 'ItemCre' fails in subscriber.

Item type X, item B, message type 'ItemCre' processes successfully in subscriber.

Item type X, item A, message type 'ItemMod' goes directly from RIB to hospital.

Item type X, item B, message type 'ItemMod' goes from RIB to subscriber.

Item type X, item A, message type 'ItemDel' goes directly from RIB to hospital.

Package name: RMSMFM_ITEMS

Spec file name: rmsmfm_itemss.pls

Body file name: rmsmfm_itemsb.pls

Package Specification – Global Variables

```
FAMILY      CONSTANT    RIB_SETTINGS.FAMILY%TYPE      'ITEM';
ITEM_ADD    CONSTANT    VARCHAR2(30)                  'itemcre';
ITEM_UPD    CONSTANT    VARCHAR2(30)                  'itemhdrmod';
ITEM_DEL    CONSTANT    VARCHAR2(30)                  'itemdel';
ISUP_ADD    CONSTANT    VARCHAR2(30)                  'itemsupcre';
ISUP_UPD    CONSTANT    VARCHAR2(30)                  'itemsupmod';
ISUP_DEL    CONSTANT    VARCHAR2(30)                  'itemsupdel';
ISC_ADD     CONSTANT    VARCHAR2(30)                  'itemsupctycre';
ISC_UPD     CONSTANT    VARCHAR2(30)                  'itemsupctymod';
ISC_DEL     CONSTANT    VARCHAR2(30)                  'itemsupctydel';
ISCD_ADD    CONSTANT    VARCHAR2(30)                  'iscdimcre';
ISCD_UPD    CONSTANT    VARCHAR2(30)                  'iscdimmod';
ISCD_DEL    CONSTANT    VARCHAR2(30)                  'iscdimdel';
```

```
UPC_ADD     CONSTANT    VARCHAR2(30)                'itemupccre';
UPC_UPD     CONSTANT    VARCHAR2(30)                'itemupcmod';
UPC_DEL     CONSTANT    VARCHAR2(30)                'itemupcdel';
BOM_ADD     CONSTANT    VARCHAR2(30)                'itembomcre';
BOM_UPD     CONSTANT    VARCHAR2(30)                'itembommod';
BOM_DEL     CONSTANT    VARCHAR2(30)                'itembomdel';
UDAF_ADD    CONSTANT    VARCHAR2(30)                'itemudaffcre';
UDAF_UPD    CONSTANT    VARCHAR2(30)                'itemudaffmod';
UDAF_DEL    CONSTANT    VARCHAR2(30)                'itemudaffdel';
UDAD_ADD    CONSTANT    VARCHAR2(30)                'itemudadatecre';
UDAD_UPD    CONSTANT    VARCHAR2(30)                'itemudadatemod';
UDAD_DEL    CONSTANT    VARCHAR2(30)                'itemudadatedel';
UDAL_ADD    CONSTANT    VARCHAR2(30)                'itemudalovcre';
UDAL_UPD    CONSTANT    VARCHAR2(30)                'itemudalovmod';
UDAL_DEL    CONSTANT    VARCHAR2(30)                'itemudalovdel';
IMG_ADD     CONSTANT    VARCHAR2(30)                'itemimagecre';
IMG_UPD     CONSTANT    VARCHAR2(30)                'itemimagemod';
IMG_DEL     CONSTANT    VARCHAR2(30)                'itemimagedel';
TCKT_ADD    CONSTANT    VARCHAR2(30)                'itemtcktcre';
TCKT_DEL    CONSTANT    VARCHAR2(30)                'itemtcktdel';
RIH_ADD     CONSTANT    VARCHAR2(30)                'relitemheadcre';
RIH_UPD     CONSTANT    VARCHAR2(30)                'relitemheadmod';
RIH_DEL     CONSTANT    VARCHAR2(30)                'relitemheaddel';
RID_ADD     CONSTANT    VARCHAR2(30)                'relitemdetcre';
RID_UPD     CONSTANT    VARCHAR2(30)                'relitemdetmod';
RID_DEL     CONSTANT    VARCHAR2(30)                'relitemdetdel';

bom_table  bom_tabtype;
empty_bom  bom_tabtype;
```

Function Level Description – ADDTOQ

```
Function: ADDTOQ
          (O_error_message          OUT    VARCHAR2,
           I_queue_rec              IN     ITEM_MFQUEUE%ROWTYPE,
           I_sellable_ind           IN     ITEM_PUB_INFO.SELLABLE_IND%TYPE,
           I_tran_level_ind         IN     ITEM_PUB_INFO.TRAN_LEVEL_IND%TYPE)
```

This public function puts an item message on ITEM_MFQUEUE for publishing to the RIB. It is called from the item trigger and the detail triggers (ITEM_SUPPLIER, ITEM_SUPP_COUNTRY, ITEM_SUPP_COUNTRY_DIM, PACKITEM, UDA_ITEM, UDA_VALUES, ITEM_IMAGE, RELATED_ITEM_HEAD, RELATED_ITEM_DETAIL). The I_queue_rec contains item and, optionally, other detail keys.

For header level insert messages (HDR_ADD), insert a record in the ITEM_PUB_INFO table. The published flag should be set to 'N'. For all message types except header level inserts (HDR_ADD), insert a record into the ITEM_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)
```

Modify the existing function as follows:

- Change the signature of this package per this specification.
- Replace the code that is in the current function with the functionality in this design.

This public procedure is called from the RIB to get the next messages. It performs a cursor loop on the unpublished records on the ITEM_MFQUEUE table (PUB_STATUS = 'U'). It should only need to execute single 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 (item). 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 and picked up again in the next loop iteration.
2. 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.
3. Get the published indicator from the ITEM_PUB_INFO table.
4. Call PROCESS_QUEUE_RECORD with the current business object.

The loop must be execute for more than one iteration in 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. The queue is locked for the current business object. This can occur because ADDTOQ, which is called from the triggers, deletes from the queue table for DTL_UPD, DTL_DEL, and HDR_DEL messages.

The information from the ITEM_MFQUEUE and ITEM_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.

If PROCESS_QUEUE_RECORD fails, the record that keeps track of which mfqueue records to delete/update should be reset. Therefore, a snapshot of the struct is taken before the call to PROCESS_QUEUE_RECORD. If the function fails, the record is reset back to the snapshot.

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 NOCOPY       RIB_BUSOBJID_TBL,
               O_routing_info       IN  OUT NOCOPY       RIB_ROUTINGINFO_TBL)
```

This public procedure performs the same tasks as GETNXT except that it only loops for a specific row in the ITEM_MFQUEUE table. The record on ITEM_MFQUEUE must match the passed in sequence number (contained in the ROUTING_INFO).

Function Level Description – PROCESS_QUEUE_RECORD (local)

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

Get relevant publishing info for the item in ITEM_PUB_INFO, including the published indicator and approved upon create indicator.

If I_hdr_published is either 'N' (not published)

- If I_hdr_published is 'N', check to see if the current message should cause the item to be published. This will be true if the status has changed to 'A'pproved or if an ITEM_SUPP_COUNTRY record has been added to an item that was approved upon create. If the item is ready to be published for the first time, the message type is a header create (HDR_ADD). If it is not ready to be published, add the record's ROWID to the structure that keeps track of ROWIDs to delete.
- Call MAKE_CREATE to build the DESC Oracle Object to publish to the RIB. This will also take care of any ITEM_MFQUEUE deletes, updating ITEM_PUB_INFO.PUBLISHED to 'Y' or 'I', and bulk updating the detail tables publish_ind column to 'Y' for those detail rows that have been published.

If the message type is an update or creates message type at any level (for example, ITEM_ADD, ISUP_ADD, ISUP_UPD, and others):

- Call RMSMFM_ITEMS_BUILD.BUILD_MESSAGE to build the DESC Oracle Object to publish to the RIB.
- RMSMFM_ITEMS_BUILD.BUILD_MESSAGE will return an indicator specifying if the record exists. The record in question is the record on the functional table corresponding to the current MFQUEUE record being processed. For example, for ITEM_ADD or ITEM_UPD message, the record exists indicator specifies whether or not the ITEM_MASTER record for the item still exists. For an ISUP_ADD or ISUP_UPD message, the record exists indicator specifies whether or not the ITEM_SUPPLIER record for the item/supplier combination still exists. If the record does not exist, the current message cannot be published.
  - If the record does not exist and the message type is an update, delete the current MFQUEUE record (that is, add the ROWID to the list of ROWIDs to be eventually deleted).
  - If the record does not exist and the message type is a create, update the current MFQUEUE record's pub_status to 'N' so that the record will be skipped but remain on the queue (that is, add the ROWID to the list of ROWIDs to be eventually updated).

If the message type is a delete message type at any level (for example, ITEM_DEL, ISUP_DEL, and others):

- Call RMSMFM_ITEMS_BUILD.BUILD_DELETE_MESSAGE to build the REF Oracle Object to publish to the RIB.
- For the current delete message, there could be a corresponding create message earlier on the queue if the create message could not be published (see update/create message type section above). If there is a corresponding create message earlier on the queue, delete both create and delete messages (that is, add the ROWIDs to the list of ROWIDs to be eventually deleted), and do not publish anything.

Finally, perform DML cleanup if a message is going to be published.

- Call UPDATE_QUEUE_TABLE to perform DML using the global record that keeps track of QUEUE records to update/delete.
- If the message type is ITEM_ADD, update the item's ITEM_PUB_INFO to published = 'Y'.
- If the message type is ITEM_DEL, delete the item's ITEM_PUB_INFO record.

Function Level Description – MAKE_CREATE (local)

This private function is used to create the Oracle Object for the initial publication of a business transaction. I_business_object contains the item header key values (item). I_rowid is the rowid of the item_mfqueue row fetched from GETNXT.

- Call BUILD_HEADER_OBJECT to get a header level Oracle Object.
- Call BUILD_DETAIL_OBJECTS to get a table of detail level Oracle objects and a table of ITEM_MFQUEUE rowids to delete with and a table of detail table rowids to update publish_ind with.
- Update ITEM_PUB_INFO.published to 'Y' or 'I' depending on if all details are published.
- Delete records from the ITEM_MFQUEUE for all rowids returned by BUILD_DETAIL_OBJECTS. Deletes are done by rowids instead of business transaction keys to ensure that nothing is deleted off the queue that has not been published.
- If the entire business transaction was added to the Oracle Object, also delete the ITEM_MFQUEUE record that was picked up by GETNXT. If the entire business transaction was **not** published, the system must leave something on the ITEM_MFQUEUE to ensure that the rest of it is picked up by the next call to GETNXT.
- Update the detail tables publish_ind column to 'Y' by each detail table of rowids returned from BUILD_DETAIL_OBJECTS.
- The header and detail level Oracle Objects are combined and returned.

Function Level Description – HANDLE_ERRORS (local)

This private procedure is called from GETNXT and PUB_RETRY when an exception is raised. I_seq_no is the sequence number of the driving ITEM_MFQUEUE record. I_function_keys contains detail level key values (item and optional detail keys).

If the error is a non-fatal error, HANDLE_ERRORS passes the sequence number of the driving ITEM_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, 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.

Package name: RMSMFM_ITEMS_BUILD

Spec file name: rmsmfm_items.pls

Body file name: rmsmfm_itemb.pls

Function Level Description – BUILD_MESSAGE

```
Function: BUILD_MESSAGE
    (O_error_msg          OUT               VARCHAR2,
     O_message            IN OUT NOCOPY     "RIB_ItemDesc_REC",
     O_rowids_rec         IN OUT NOCOPY     ROWIDS_REC,
     O_record_exists      IN OUT            BOOLEAN,
     I_message_type       IN                ITEM_MFQUEUE.MESSAGE_TYPE%TYPE,
     I_tran_level_ind     IN                ITEM_PUB_INFO.TRAN_LEVEL_IND%TYPE,
     I_queue_rec          IN                ITEM_MFQUEUE%ROWTYPE)
```

The private function is responsible for building detail level DESC Oracle Objects. It builds as many detail Oracle Object as it can given the passed in message type and business object keys (item).

Call the following:

- BUILD_HEADER_DETAIL
- BUILD_SUPPLIER_DETAIL
- BUILD_COUNTRY_DETAIL
- BUILD_DIM_DETAIL
- BUILD_UDA_LOV_DETAIL
- BUILD_UDA_FF_DETAIL
- BUILD_UDA_DATE_DETAIL
- BUILD_IMAGE_DETAIL
- BUIILD_UPC_DETAIL
- BUILD_BOM_DETAIL
- BUILD_TICKET_DETAIL
- BUILD_RELATED_ITEMS_HEAD
- BUILD_RELATED_ITEMS_DETAIL (The object built in this function will be a child of the object built in the BUILD_ RELATED_ITEMS_HEAD function based on the relationship_id)

Function Level Description – BUILD_DELETE_MESSAGE

```
Function: BUILD_DETAIL_CHANGE_OBJECTS
           (O_error_msg       OUT             VARCHAR2,
            O_message         IN OUT NOCOPY   "RIB_ItemDesc_REC",
            I_message_type    IN              ITEM_MFQUEUE.MESSAGE_TYPE%TYPE,
            I_business_obj    IN              ITEM_KEY_REC)
```

This function builds a REF Oracle Object to publish to the RIB for all delete message types (for example, ITEM_DEL, ISUP_DEL, ISC_DEL, and others).

The function also checks to see if there is a corresponding Create message for the current delete message. If so, O_create_rowid is set. This is used to determine if the Delete message should be published (see PROCESS_QUEUE_RECORD description above). If both Create and Delete messages are on the queue, neither are published.

Detail creates and detail update messages (DTL_ADD, DTL_UPD). I_business_obj contains the header level key values (item).

Function Level Description – BUILD_HEADER_OBJECT (local)

This private function accepts item header key values (item), builds and returns a header level DESC Oracle Object. Call GET_ITEM_INFO to retrieve data supplementary to ITEM_MASTER. If the item is not found on ITEM_MASTER, O_record_exists is set to FALSE.

Function Level Description – BUILD DETAIL functions (all local)

The following functions have the same format:

- BUILD_SUPPLIER_DETAIL
- BUILD_COUNTRY_DETAIL
- BUILD_DIM_DETAIL
- BUILD_UDA_LOV_DETAIL

- BUILD_UDA_FF_DETAIL
- BUILD_UDA_DATE_DETAIL
- BUILD_IMAGE_DETAIL
- BUIILD_UPC_DETAIL
- BUILD_BOM_DETAIL
- BUILD_TICKET_DETAIL
- BUILD_RELATED_ITEMS_HEAD
- BUILD_RELATED_ITEMS_DETAIL

They have the same specifications, except as noted below.

The functions for building detail nodes for the ITEMDESC message work in the same way. The functions build as many detail Oracle Objects as they can, given the passed in message type and business object keys.

The difference between the different detail functions lies in the data being accessed. BUILD_SUPPLIER_DETAIL retrieves information from ITEM_SUPPLIER, BUILD_COUNTRY_DETAIL retrieves information from ITEM_SUPP_COUNTRY, etc.

BUILD_SUPPLIER_DETAIL and BUILD_COUNTRY_DETAIL are the only functions that have the input parameter I_orderable_item. This is used to validate orderable items. If an item is orderable, and the initial ITEM_ADD message is being created, at least one supplier node and one supplier/country node are required. This is the only business validation done by the item publisher.

The BUILD_ RELATED_ITEMS_HEAD function retrieves data (item relationship details) from the RELATED_ITEM_HEAD table and builds detail nodes for the ITEMDESC message. Each of these detail nodes has child nodes if the item relationship contains related items records in the RELATED_ITEM_DETAIL table. These child nodes are built by the BUILD_ RELATED_ITEMS_DETAIL function which is called within the BUILD_ RELATED_ITEM_HEAD function. These child nodes are optional for the detail nodes.

If the original create message is being published (I_message_type would be ITEM_ADD)

- Select all detail records for the business transaction. Return a table of ITEM_MFQUEUE rowids for each message that is placed into the Oracle Object.
- Since the message being published is ITEM_ADD, there may not be a record on the MFQUEUE table for each detail record that needs to be retrieved. Therefore, no inner join to the MFQUEUE table is done. However, if there are any MFQUEUE records for details, they should be deleted. Therefore, a UNION to a second query is done to select all relevant MFQUEUE records for deletion.

If the message being published is a detail add or detail update (for example, ISUP_ADD, ISUP_UPD, ISC_ADD, ISC_UPD)

- Select all detail records for the business transaction. Return a table of ITEM_MFQUEUE rowids for each message that is placed into the Oracle Object.
- Since the message being published is a detail create or update, the only details that should be added to the message are those details that have a record on the MFQUEUE table. Therefore, an inner join between the MFQUEUE table and the business detail table is performed. Any MFQUEUE records retrieved will have their ROWIDs added to the list of ROWIDs that will eventually be deleted.
- If no records are retrieved for the detail record query, O_records_exist is set to FALSE.

A concern here is making sure that the system does not delete information from the queue table that has not been published. For this reason, the system does deletes by ROWID. The system also tries to get everything in the same cursor to ensure that the

message published matches the deletes that are performed from the ITEM_MFQUEUE table regardless of trigger execution during GETNXT calls.

Function Level Description – GET_ITEM_INFO (local)

This private function gets ITEM_MASTER as input and retrieves supplementary data. For example, each item has a department, class, and subclass. GET_ITEM_INFO will retrieve the descriptions for these three fields. This function is called from BUILD_HEADER_OBJECT.

Function Level Description – BUILD_DIMENSION_DESCRIPTIONS (local)

This private function is similar to GET_ITEM_INFO in that it retrieves supplementary data. This function, however, is called when item/supplier/country/dimension message nodes are being populated. This function is called from BUILD_DIM_DETAIL.

Trigger Impact

Trigger name: EC_TABLE_IEM_AIUDR.TRG (mod)

Trigger file name: ec_table_iem_aiudr.trg (mod)

Table: ITEM_MASTER

Modify the trigger on the ITEM table to capture Inserts, Updates, and Deletes. Remove all of the code except the code that checks the item_level and tran_level. This is needed to determine which message type to send to the queue, item or UPC (reference item).

- **Inserts**: Send the header level item info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.ITEM_ADD or RMSMFM_ITEM.UPC_ADD.
- **Updates**: Send the header level item info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.ITEM_UPD or RMSMFM_ITEM.UPC_UPD.
- **Deletes**: Send the header level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.ITEM_DEL or RMSMFM_ITEM.UPC_DEL.

In all these cases, build the function keys for ADDTOQ with item.

Trigger name: EC_TABLE_ISP_AIUDR.TRG (mod)

Trigger file name: ec_table_isp_aiudr.trg (mod)

Table: ITEM_SUPPLIER

Populate the ITEM_MFQUEUE table according to the message type. Make sure that only transaction level items are added to the ITEM_MFQUEUE table.

- **Inserts**: Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_ADD.
- **Updates**: Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_UPD.
- **Deletes**: Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_DEL.

In all these cases, build the function keys for ADDTOQ with item and supplier.

Trigger name: EC_TABLE_ISC_AIUDR.TRG (mod)

Trigger file name: ec_table_isc_aiudr.trg (mod)

Table: ITEM_SUPP_COUNTRY

Populate the ITEM_MFQUEUE table according to the message type. Make sure that only transaction level items are added to the ITEM_MFQUEUE table.

- **Inserts**: Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_ADD.
- **Updates**: Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_UPD.
- **Deletes**: Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_DEL.

In all these cases, build the function keys for ADDTOQ with item, supplier and origin_country_id.

Trigger name: EC_TABLE_ISD_AIUDR.TRG (mod)

Trigger file name: ec_table_isd_aiudr.trg (mod)

Table: ITEM_SUPP_COUNTRY_DIM

Populate the ITEM_MFQUEUE table according to the message type. Make sure that only transaction level items are added to the ITEM_MFQUEUE table.

- **Inserts:** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_ADD.
- **Updates:** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_UPD.
- **Deletes:** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_DEL.

In all these cases, build the function keys for ADDTOQ with item, supplier, origin_country_id.

Trigger name: EC_TABLE_PKS_AIUDR.TRG (mod)

Trigger file name: ec_table_pks_aiudr.trg (mod)

Table: PACKITEM_BREAKOUT

This trigger captures inserts, updates and deletes on the table. It populates a PL/SQL table of records, RMSMFM_ITEMS.BOM_TABLE, which will be used in the statement trigger to build an XML message and place it on the item queue.

Trigger name: EC_TABLE_PKS_IUDS.TRG (mod)

Trigger file name: ec_table_pks_aiudr.trg (mod)

Table: PACKITEM_BREAKOUT

This trigger will group all of the data currently stored in the PL/SQL table of records populated by the EC_TABLE_PKS_AIUDR trigger, and call RMSMFM_ADDTOQ for every pack component in the table of records.

Trigger name: EC_TABLE_UIT_AIUDR.TRG (mod)

Trigger file name: ec_table_uit_aiudr.trg (mod)

Table: UDA_ITEM_DATE

Populate the ITEM_MFQUEUE table according to the message type. Make sure that only transaction level items are added to the ITEM_MFQUEUE table.

- **Inserts:** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_ADD.
- **Updates:** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_UPD.
- **Deletes:** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_DEL.

In all these cases, build the function keys for ADDTOQ with item, uda_id.

Trigger name: EC_TABLE_UIF_AIUDR.TRG (mod)

Trigger file name: ec_table_uif_aiudr.trg (mod)

Table: UDA_ITEM_FF

Populate the ITEM_MFQUEUE table according to the message type. Make sure that only transaction level items are added to the ITEM_MFQUEUE table.

- **Inserts:** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_ADD.
- **Updates:** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_UPD.
- **Deletes:** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_DEL.

In all these cases, build the function keys for ADDTOQ with item, uda_id.

Trigger name: EC_TABLE_UIL_AIUDR.TRG (mod)

Trigger file name: ec_table_uil_aiudr.trg (mod)

Table: UDA_ITEM_LOV

Populate the ITEM_MFQUEUE table according to the message type. Make sure that only transaction level items are added to the ITEM_MFQUEUE table.

- **Inserts;** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_ADD.
- **Updates:**Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_UPD.
- **Deletes:** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_DEL.

In all these cases, build the function keys for ADDTOQ with item, uda_id and uda_value.

Trigger name: EC_TABLE_RIH_AIUDR.TRG (mod)

Trigger file name: ec_table_rih_aiudr.trg (mod)

Table: RELATED_ITEM_HEAD

Populate the ITEM_MFQUEUE table according to the message type. Make sure that only transaction level items are added to the ITEM_MFQUEUE table.

- **Inserts:** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_ADD.
- **Updates:** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_UPD.
- **Deletes:** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_DEL.

In all these cases, build the function keys for ADDTOQ with item and relationship_id.

Trigger name: EC_TABLE_RID_AIUDR.TRG (mod)

Trigger file name: ec_table_rid_aiudr.trg (mod)

Table: RELATED_ITEM_DETAIL

Populate the ITEM_MFQUEUE table according to the message type. Make sure that only transaction level items are added to the ITEM_MFQUEUE table.

- **Inserts:** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_ADD.
- **Updates:** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_UPD.
- **Deletes:** Send the detail level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ITEM.DTL_DEL.

In all these cases, build the function keys for ADDTOQ with item, relationship_id and related_item.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
itemcre	Item Create Message	ItemDesc.xsd
itemmod	Item Modify Message	ItemDesc.xsd
itemdel	Item Delete Message	ItemRef.xsd
itemsupcre	Item Supplier Create Message	ItemSupDesc.xsd
itemsupmod	Item Supplier Modify Message	ItemSupDesc.xsd
itemsupdel	Item Supplier Delete Message	ItemSupRef.xsd
itemsupctycre	Item Supplier Country Create Message	ItemSupCtyDesc.xsd
itemsupctymod	Item Supplier Country Modify Message	ItemSupCtyDesc.xsd
itemsupctydel	Item Supplier Country Delete Message	ItemSupCtyRef.xsd

Message Types	Message Type Description	XML Schema Definition (XSD)
iscdimcre	Item Supplier Country Dimension Create Message	ISCDimDesc.xsd
iscdimmod	Item Supplier Country Dimension Modify Message	ISCDimDesc.xsd
iscdimdel	Item Supplier Country Dimension Delete Message	ISCDimRef.xsd
itemupccre	Item UPC Create Message	ItemUPCDesc.xsd
itemupcmod	Item UPC Modify Message	ItemUPCDesc.xsd
itemupcdel	Item UPC Delete Message	ItemUPCRef.xsd
itembomcre	Item BOM Create Message	ItemBOMDesc.xsd
itembommod	Item BOM Modify Message	ItemBOMDesc.xsd
itembomdel	Item BOM Delete Message	ItemBOMRef.xsd
itemudaffcre	Item UDA Free Form TextCreate Message	ItemUDAFFDesc.xsd
itemudaffmod	Item UDA Free Form Text Modify Message	ItemUDAFFDesc.xsd
itemudaffdel	Item UDA Free Form Text Delete Message	ItemUDAFFRef.xsd
itemudalovcre	Item UDA LOV Create Message	ItemUDALOVDesc.xsd
itemudalovmod	Item UDA LOV Modify Message	ItemUDALOVDesc.xsd
itemudalovdel	Item UDA LOV Delete Message	ItemUDALOVRef.xsd
itemudadatecre	Item UDA Date Create Message	ItemUDADateDesc.xsd
itemudadatemod	Item UDA Date Modify Message	ItemUDADateDesc.xsd
itemudadatedel	Item UDA Date Delete Message	ItemUDADateRef.xsd
itemimagecre	Item Image Create Message	ItemImageDesc.xsd
itemimagemod	Item Image Modify Message	ItemImageDesc.xsd
itemimagedel	Item Image Delete Message	ItemImageRef.xsd
relitemheadcre	Item Relationship Create Message	RelatedItemDesc.xsd
relitemheadmod	Item Relationship Modify Message	RelatedItemDesc.xsd
relitemheaddel	Item Relationship Delete Message	RelatedItemRef.xsd
relitemdetcre	Related Item Create Message	RelatedItemDesc.xsd
relitemdetmod	Related Item Modify Message	RelatedItemDesc.xsd
relitemdetdel	Related Item Delete Message	RelatedItemRef.xsd

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_MFQUEUE	Yes	Yes	Yes	Yes
ITEM_PUB_INFO	Yes	Yes	Yes	Yes
ITEMLOC_MFQUEUE	Yes	No	No	Yes
ITEM_MASTER	Yes	No	No	No
ITEM_SUPPLIER	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
ITEM_SUPP_COUNTRY_DIM	Yes	No	No	No
UDA_ITEM_LOV	Yes	No	No	No
UDA_ITEM_DATE	Yes	No	No	No
UDA_ITEM_FF	Yes	No	No	No
ITEM_IMAGE	Yes	No	No	No
PACKITEM_BREAKOUT	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
ITEM_TICKET	Yes	No	No	No
CODE_DETAIL	Yes	No	No	No
DEPS	Yes	No	No	No
CLASS	Yes	No	No	No
SUBCLASS	Yes	No	No	No
V_DIFF_ID_GROUP_TYPE	Yes	No	No	No
ITEM_ZONE_PRICE	Yes	No	No	No
PACKITEM	Yes	No	No	No
RELATED_ITEM_HEAD	Yes	No	No	No
RELATED_ITEM_DETAIL	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 (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.

Item Location Publication API

Functional Area

Foundation

Business Overview

RMS defines and publishes item-location relationships. The details about item-location relationship creation, updation and de-activation are important for other systems for smooth functioning of several business processes. For example, when an new item-location relationship is created, the Point-Of-Sale system needs to be made aware of this information so that it can smoothly process subsequent sales and return activities at the Point-of-sale etc. The purpose of this API is to publish such information to be subscribed and consumed by other systems.

Package Impact

As and when item-location relationships are created or modified as part of various business processes, such events are captured as using triggers on the item location set of tables. The trigger then invokes methods from this API to successfully publish the captured information.

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,
     I_mult_runs_per_day_ind   IN      REPL_ITEM_LOC.MULT_RUNS_PER_DAY_IND%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').

Because ITEMLOC records should not be published before ITEM records a clause is included in the driving cursor that checks for ITEM CREATE messages on the ITEM_MFQUEUE table. The ITEMLOC_MFQUEUE record will not be selected from the driving cursor if the ITEM CREATE message still exists on ITEM_MFQUEUE. Also, ITEMLOC_MFQUEUE cleanup is 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 also deletes from ITEMLOC_MFQUEUE when a DELETE message for a non-published item is encountered.

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 raises 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 builds the Oracle Object message to pass back to the RIB. If PROCESS_QUEUE_RECORD does not run successfully, GETNXT will raise 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            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)

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) the function will 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) the function will 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)

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.

- Selects all records on the ITEMLOC_MFQUEUE that are for the same item. The records are fetched in order of seq_no on the MFQUEUE table. The records are fetchee into a table using BULK COLLECT, with MAX_DETAILS_TO_PUBLISH as the LIMIT clause.
- The records in the BULK COLLECT table are looped through. If the record's message_type differs from the message type passed into the function, it will exit from the loop. Otherwise, it will add the data from the record to the Oracle Object being used for publication. If the input message type is not REPL_UPD then the Purchase Type for the item's department is retrieved and it is added to the oracle object.
- Ensures that ITEMLOC_MFQUEUE is deleted from as needed.
- Ensures 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 the system does not delete records from the queue table that have not been published. For this reason, the system performs deletes by ROWID. The system will also get everything in the same cursor. This should ensure that the message published matches the deletes performed from the ITEMLOC_MFQUEUE table regardless of trigger execution during GETNXT calls.

Function Level Description – BUILD_DETAIL_DELETE_OBJECTS (local)

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)

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, 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

A trigger exists on the ITEM_LOC to capture inserts, updates, and deletes.

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

Trigger name: EC_TABLE_ITL_AIUDR.TRG (mod)

Trigger file name: ec_table_itl_aiudr.trg (mod)

Table: ITEMLOC

- **Inserts:** Sends 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:** Sends 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:** Sends 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 will 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 table REPL_ITEM_LOC to capture inserts, updates, and deletes.

Updates:

- Sends 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**: Sends 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.

Message XSD

Below are the filenames that correspond with each message type. Please consult the RIB documentation for each message type in order to get a detailed picture of the composition of each message.

Message Types	Message Type Description	XML Schema Definition (XSD)
ItemLocCre	Item Loc Create Message	ItemLocDesc.xsd
ItemLocMod	Item Loc Modify Message	ItemLocDesc.xsd
ItemLocDel	Item Loc Delete Message	ItemLocRef.xsd
ItemLocReplMod	Item Loc Replenishment Modify Message	ItemLocDesc.xsd

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
DEPS	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 (at most 2-3 seconds). It is also assumed that this will occur rarely because 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 Publication API

Functional Area

Foundation Data

Business Overview

This API publishes information regarding all the levels of the merchandise hierarchy to the RIB such that all the downstream applications may subscribe to it and have merchandise hierarchy information in sync with RMS.

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 divisons, the business object ID will be the division. 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.

File name: rmsmfm_merchhiers/b.pls

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' - Hospital. 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)

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. 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)

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' -Hospital to the RIB as well. It then updates the status of the queue record to 'H' 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' – Error 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' to 'E'.

Message XSD

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

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

Design Assumptions

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

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

Order Publication API

Functional Area

Purchase Orders

Business Overview

Purchase order (PO) functionality in RMS consists of order messages published to the Oracle Retail Integration Bus (RIB), and batch modules that internally process purchase order data and uploads EDI transmitted order. This overview describes how both order messages and batch programs process this data.

Creating of Purchase Orders

A purchase order is created using the following:

- Through online using the ordering dialog.
- Replenishment processes.
- When the supplier contract type is 'B'.
- By a supplier, in a vendor managed inventory environment.
- Direct store delivery (defined as delivery of merchandise or a service that does not result from the prior creation of a PO). For more information, see *Oracle Retail Merchandising System Operations Guide, Volume 1 - Batch Overviews and Designs, Chapter Purchase Order.*
- Buyer Worksheet dialog.
- Truck splitting.
- Customer Order webservice/RIB.
- Franchise Order.

Purchase Order Messages

After purchase orders are published to the RIB, the following associated activity occurs:

- Work orders associated with items on the PO are published to the RIB through the work order message process.
- An allocation (also known as pre-distribution) of items on the PO is published to the RIB through the stock order message process.
- A PO can be closed only after all appointments against the purchase order are closed. A closed appointment indicates that all merchandise has been received. RMS subscribes to appointment messages from the RIB.
- 'Version' refers to any change to a purchase order by a retailer's buyer; whereas 'Revision' refers to any change to a purchase order initiated by a supplier.

Order Message Processes

RMS publishes two sets of PO messages to RIB for two kinds of subscribing applications. The first set of messages contains only virtual locations in RMS. Applications that understand virtual locations subscribe to these messages.

RMS publishes a second set of PO messages for applications that can subscribe only to conventional, physical location data, such as a Warehouse Management System.

Ordering publication is primarily based off of the ORDHEAD, ORDSKU, and ORDLOC tables.

ORDHEAD is the parent table containing high level ordering information such as what supplier is being ordered from, when the order must take place, and so on. ORDSKU is a child of ORDHEAD and contains the item(s) that are ordered, the size of the pack being ordered.

ORDLOC is a child of ORDSKU that contains the location(s) each item on the order is going to and how much of each item is ordered. Based on this table hierarchy, two levels of messages exist for order publishing. A header message is primarily driven off of the ORDHEAD table and the detail message that is primarily driven off both the ORDSKU and ORDLOC tables.

If the purchase order is a customer order (order_type = 'CO' with a stockholding store), the Customer Order Number and Fulfillment Order Number retrieved from the ORDCUST table will be included in the header message and published.

Each message level contains three types of messages; Create, Modify, and Delete. The 'POCre' or 'POHdrMod' message is created when an insertion or modification to the ORDHEAD table is made respectively. The 'PODel' message is created when an order is deleted from the ORDHEAD table. 'PODtlCre' or 'PODtlMod' message is created when a record is inserted or modified on the ORDLOC table respectively. 'PODtlDel' is created when an ORDLOC record is deleted.

Package Impact

Create a Worksheet Order

1. **Prerequisites**: Orders are created through various methods. Orders created manually by a user, through a replenishment process (order can be created in either worksheet or approved status), uploaded from a vendor, through a contract, through customer order creation or through a franchise order creation.
2. **Activity Detail**: At this point, the order is not seen externally from RMS.
3. **Messages**: When the order is created, a header message 'POCre' is written to the ordering queue table. Upon detail additions, each will have a 'PODtlCre' message

written to the ordering queue. Ordering messages are added, updated, and removed from the queue as the order is modified prior to approval.

Modify Pre-Approved

1. **Prerequisites**: Order is still in worksheet status and has not been approved and is set back to worksheet.
2. **Activity Detail**: At this point, items are modified, added or removed from the order. The order is split, scaled, and rounded in addition to having deals, brackets applied.
3. **Messages**: Each change causes a 'POHdrMod' or 'PODtlMod' message. These messages replaces previous create messages if there was a modification, delete a previous message if there was a delete, or add a new message to the queue for inserts.

Approve

1. **Prerequisites**: Line items must exist for the order to be approved. Relevant dates (not before, not after, pickup) must exist, plus certain other business validation rules based on system options.
2. **Activity Detail**: At this point, the order is initially approved which means external systems will now have constant visibility to all ordering transactions. The user can no longer delete line items: Instead, they are cancelled. Canceling decrements the order quantity by amount already received.
3. **Messages**: The approval message sets an indicator signifying the approval creates message must be built. This is a hierarchical snapshot synchronous message built in the family manager by attaching all of the 'PODtlDesc' messages with the 'POHdrDesc' message creates a 'POCre' message.

Modify in 'A' status

1. **Prerequisites**: Order must be currently approved.
2. **Activity Detail**: Numerous fields at the header level (none at the detail level) can be changed while the order is approved. This change creates a message.
3. **Messages**: A 'POHdrMod' message is created for order at the end of the session the order was modified. This message is published immediately as the order is already been published. If the order has not been published, then this message follows the create message sent out.

Redistribute

1. **Prerequisites**: Order must be in approved or worksheet status. Order must not be a contract order. No shipments/appointments may exist against the order. Items with allocations cannot be redistributed.
2. **Activity Detail**: User chooses which items to redistribute. Each chosen details are removed from the order. This creates delete messages for each one. A new location is then chosen to redistribute the items to. Each item/location record creates a message. Note that if user chooses to redistribute records, then cancels out of redistribution, delete and create messages for the chosen records is inserted into the queue even though no changes were actually made online.
3. **Messages**: A 'PODtlDel' message is created for each item/location removed from the order. If the order has not yet been approved, then these messages removes previous create messages. For already approved orders, then a message is published. For each redistributed item, a 'PODtlCre' message is created.

Unapprove

1. **Prerequisites**: Order must currently be in approved status. Shipments/Appoinments may exist against the order.
2. **Activity Detail**: This changes the status of the order back to worksheet. This creates a message. Existing details is modifiable. New records may be added to the order. Items may not be deleted from the order. However, the order quantity of the items can be canceled down to the received or appointment expected quantity.
3. **Messages**: A 'POHdrMod' message is created for order at the end of the session the order was modified. This message is published immediately as the order is already have been published. If the order has not been published, then this message follows the create message sent out.

Modify

1. **Prerequisites**: Order must be in worksheet status and have already been approved.
2. **Activity Detail**: If modification occurs at the header level, a header message is created. A detail message is created for each modified or added detail record. Detail records cannot be deleted; only their quantities can be canceled.
3. **Message**: A 'POHdrMod' message is created for order at the end of the session if the header was modified. A 'PODtlCre' or 'PODtlMod' message is created for each detail record added or modified respectively.

Close

1. **Prerequisites**: Order must currently be in an approved status or in worksheet status and which is already approved. No outstanding shipments/appointments may exist against any line items of the order.
2. **Activity Detail**: The status changes to closed. This creates a message. Any outstanding unreceived quantity is canceled out. No detail is modifiable while the order is in this status.
3. **Message**: A 'POHdrMod' message is created for order at the end of the session the order was modified. A 'PODtlMod' message is created for each line item that had outstanding un-received quantity. These messages are published immediately as the order is already published. If the order has not been published, then this message follows the create message sent out.

Reinstate

1. **Prerequisites**: Order must be in closed status. Orders that have been fully received (closed through receiving dialogue) cannot be reinstated.
2. **Activity Detail**: The status changes to worksheet. This creates a header level message. All canceled quantities is added back to order quantities. Details are modifiable.
3. **Message**: A 'POHdrMod' message is created for order at the end of the session the order was modified. A 'PODtlMod' message is created for each line item that had outstanding canceled quantity. These messages are published immediately as the order is already published. If the order are not published, then this message follows the create message sent out.

Delete

1. **Prerequisites**: If the user deletes the order manually, then the order needs to be in worksheet status and never been approved. Else, for approved orders, the following explanation details the business validation for deleting orders. If the import indicator on the SYSTEM OPTIONS table (import_ind) is 'N' and if invoice matching is not installed, then all details associated with an order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months). If invoice matching is installed, then all details associated with an order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months). Orders are deleted only if shipments from the order have been completely matched to invoices or closed, and all those invoices have been posted. If the import indicator on the SYSTEM OPTIONS table (import_ind) is 'Y' and if invoice matching is not installed, then all details associated with the order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months) , as long as all ALC records associated with an order are in 'Processed' status, specified in ALC_HEAD (status). If invoice matching is installed, then all details associated with an order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months), as long as all ALC records associated with an order are in 'Processed' status, specified in ALC_HEAD (status), and as long as all shipments from the order have been completely matched to invoices or closed, and all those invoices have been posted.
2. **Activity Detail**: Deleting orders will create a message for each detail attached to the order plus the header record.
3. **Messages**: If the order has not been approved, then the 'PODel' and 'PODtlDel' messages created will remove all the previous messages on the ordering queue table. If the order has been approved, then a 'PODtlDel' message will be created for each detail record and a 'PODel' message for the header.

Filename: rmsmfm_orderb.pls

Function Level Description – ADDTOQ

```
Function: ADDTOQ
           (O_error_message          OUT     VARCHAR2,
            I_message_type           IN      ORDER_MFQUEUE.MESSAGE_TYPE%TYPE,
            I_order_no               IN      ORDHEAD.ORDER_NO%TYPE,
            I_order_type             IN      ORDHEAD.ORDER_TYPE%TYPE,
            I_order_header_status    IN      ORDHEAD.STATUS%TYPE,
            I_supplier               IN      ORDHEAD.SUPPLIER%TYPE,
            I_item                   IN      ORDLOC.ITEM%TYPE,
            I_location               IN      ORDLOC.LOCATION%TYPE,
            I_loc_type               IN      ORDLOC.LOC_TYPE%TYPE,
            I_physical_location      IN      ORDLOC.LOCATION%TYPE)
```

This procedure is called by either the ORDHEAD or ORDLOC row trigger, and takes the message type, table primary key values (order_no for ORDHEAD table and order_no, item, location (virtual) and physical location for ORDLOC table) and the message itself. It inserts a row into the message family queue along with the passed in values and the next sequence number from the message family sequence. The pub status will always be 'U' except for PO create messages, then it will be 'N'. The approve indicator will always be 'N' except when the order is approved for the first time, then it will be 'Y'. It returns error codes and strings according to the standards of the application in which it is being implemented.

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)
```

This publicly exposed procedure is typically called by a RIB publication adaptor. Its parameters are well defined and arranged in a specific order. The message type is the RIB defined short message name, the message is the xml message, and the family key(s) (order_no for ORDHEAD table and order_no, item, location (virtual) and physical location for ORDLOC table) are the key for the message as pertains to the family, not all of which will necessarily be populated for all message types.

This program loops through each message on the ORDER_MFQUEUE table, and calls PROCESS_QUEUE_RECORD. When no messages are found, the program exits returning the 'N'o message found API code.

The error text parameter contains application-generated information, such as the application's sequence number of the message that failed, and the Oracle or other error that occurred when the retrieval failed.

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

Function Level Description – PROCESS_QUEUE_RECORD (local)

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.

Check to see if the business object is being published for the first time. If the published_ind on the pub_info table is 'N', then it is not yet published.

If the record from ORDER_MFQUEUE table is a header delete (HDR_DEL) and published_ind is 'N'.

- Delete the record from the pub info table.
- Call DELETE_QUEUE_REC.

If the record from ORDER_MFQUEUE table is a header delete (HDR_DEL).

- Build and pass the RIB_PORef_REC object.
- Call GET_ROUTING_TO_LOCS.
- Delete the record from the pub info table.
- Delete the record from the order_details_published table.
- Call DELETE_QUEUE_REC.

If the published_ind is 'N' or 'I'.

- If the publish_ind is 'N' call MAKE_CREATE with the message_type 'HDR_ADD'.
- Otherwise, call MAKE_CREATE with the message_type 'DTL_ADD'.

If the record from ORDER_MFQUEUE table is a header update (HDR_UPD).

- Call BUILD_HEADER_OBJECT.
- Update order_pub_info by setting the published indicator to 'Y'.
- Call GET_ROUTING_TO_LOCS.
- Call DELETE_QUEUE_REC.

If the record from ORDER_MFQUEUE table is a detail insert (DTL_ADD) or detail update (DTL_UPD).

- Call BUILD_DETAIL_CHANGE_OBJECTS.
- If the record from ORDER_MFQUEUE table is a detail delete (DTL_DEL).
- Call BUILD_DETAIL_DELETE .
- Call ROUTING_INFO_ADD.

Function Level Description – MAKE_CREATE (local)

This function is used to create the Oracle Object for the initial publication of a business transaction.

- Call BUILD_HEADER_OBJECT to get a header level Oracle Object plus any extra functional holders.
- Call BUILD_DETAIL_OBJECTS to get a table of detail level Oracle objects and a table of ORDER_MFQUEUE rowids to delete.
- Use the header level Oracle Object and functional holders to update the ORDER_PUB_INFO.
- Delete records from the ORDER_MFQUEUE for all rowids returned by BUILD_DETAIL_OBJECTS. Deletes are done by rowids instead of business transaction keys to ensure that nothing is deleted off the queue that has not been published.
- If the entire business transaction was added to the Oracle Object, also delete the ORDER_MFQUEUE record that was picked up by GETNXT. If the entire business transaction was not published we need to leave something on the ORDER_MFQUEUE to ensure that the rest of it is picked up by the next call to GETNXT.
- The header and detail level Oracle Objects are combined and returned.

Function Level Description – BUILD_HEADER_OBJECT (local)

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

Call GET_MSG_HEADER.

Function Level Description – BUILD_DETAIL_OBJECTS (local)

The function is responsible for building detail level Oracle Objects. It builds as many detail Oracle Object as it can given the passed in message type and business object keys.

If the function is being called from MAKE_CREATE:

- Select any unpublished detail records from the business transaction (use an indicator on the functional detail table itself or ORDER_DETAILS_PUBLISHED). Create Oracle Objects for details that are selected by calling BUILD_SINGLE_DETAIL.

If the function is not being called from MAKE_CREATE:

- Select any details on the ORDER_DETAILS_PUBLISHED that are for the same business transaction and for the same message type. Create Oracle Objects for details that are selected by calling BUILD_SINGLE_DETAIL.

Create other necessary Oracle objects and insert into and update the ORDER_DETAILS_PUBLISHED table for details that were published.

Function Level Description – BUILD_SINGLE_DETAIL (local)

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

Function Level Description – BUILD_DETAIL_CHANGE_OBJECTS (local)

Either pass in a header level Oracle Object or call BUILD_HEADER_OBJECT to build one.

Call BUILD_SINGLE_DETAIL to get the delete level Oracle Objects.

Perform any BULK DML statements given the output from BUILD_ DETAIL_OBJECTS

Build any ROUTING_INFO as needed.

Function Level Description – BUILD_DETAIL_DELETE (local)

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

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

Perform any BULK DML statements for deletion from ORDER_MFQUEUE and ORDER_DETAILS_PUBLISHED.

Call BUILD_DETAIL_DELETE_WH for Warehouses.

Function Level Description – DELETE_QUEUE_REC (local)

Delete the passed in data from the queue table.

Function Level Description – BUILD_DETAIL_DELETE_WH (local)

Builds Oracle objects based on the records found in the queue table that are from the ORDLOC table.

Function Level Description – ROUTING_INFO_ADD (local)

Build any ROUTING_INFO.

Function Level Description – GET_ROUTING_TO_LOCS (local)

Build the ROUTING_INFO by adding locations.

Function Level Description – GET_MSG_HEADER (local)

Perform any lookups to complete the header information.

Function Level Description – LOCK_THE_BLOCK (local)

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 the queue table for DTL_UPD, DTL_DEL, and HDR_DEL messages.

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_seq_no              IN             order_mfqueue.seq_no%TYPE,
        I_order_no            IN             order_mfqueue.order_no%TYPE,
        I_item                IN             order_mfqueue.item%TYPE,
        I_physical_location   IN             order_mfqueue.physical_location%TYPE,
        I_loc_type            IN             order_mfqueue.loc_type%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 ORDER_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.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
POCre	Purchase Order Create Message	PODesc.xsd
POHdrMod	Purchase Order Modify Message	PODesc.xsd
PODel	Purchase Order Delete Message	PORef.xsd
PODtlCre	Purchase Order Detail Create Message	PODesc.xsd
PODtlMod	Purchase Order Detail Modify Message	PODesc.xsd
PODtlDel	Purchase Order Detail Delete Message	PORef.xsd

Design Assumptions

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

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ORDHEAD	Yes	No	No	No
ORDLOC	Yes	No	No	No
ORDSKU	Yes	No	No	No
ORDER_MFQUEUE	Yes	Yes	Yes	Yes
ORDER_PUB_INFO	Yes	Yes	Yes	Yes
ORDER_DETAILS_PUBLISHED	Yes	Yes	Yes	Yes

Partner Publication API

Functional Area

Foundation Data

Business Overview

RMS publishes data about partners in messages to Retail Integration Bus (RIB). Other application that needs to keep their partner synchronized with RMS subscribe to these messages.

External Finishers

External finishers are created as partners in RMS, and given the Partner Type 'E', indicating that the partner is an External finisher. Once a new external finisher is set up in RMS, a trigger on the partner table adds the external finisher to a new queue table. Information on that table is published via the RIB. A conversion of this RIB message converts the external finisher to a 'Location' so that it can be consumed by the location APIs of external systems such as RWMS.

RWMS and other integration subsystems subscribe to the external finisher through their location subscription APIs. A RIB TAFR parses the partner messages of partner type 'E' and returns location attributes for RWMS and other integration subsystems to subscribe to. RMS ensures that there will never be duplicates among the partner ID, store ID and warehouse ID.

The RWMS transfer subscription process does not check for location types. As a result, transfers involving an external finisher are treated like any other location types.

To facilitate the routing of external finisher and primary address of the primary address type, header level routing info will contain the name of 'partner_type' with value 'E'. Detail level routing info will contain the name of 'primary_addr_type_ind' with value of

'Y' or 'N' and the name of 'primary_addr_ind' with value of 'Y' or 'N'. This will allow the RIB to route the external finishers and their addresses to the correct applications.

RMS will publish to the RIB create, mod and delete messages of partners along with their multiple addresses via a partner publishing message.

The insert/update/delete on the partner table and the addr table with module 'PTNR' (for partner) will be published. The output message will be in hierarchical structure, with partner information at the header level and the address information at the detail level. Because this is a low volume publisher, multi-threading capability is not supported. In addition, the system assumes that it only needs to publish the current state of the partner, not every change.

If multiple addresses are associated with a partner, this publisher is designed with the assumption that RWMS and other integration subsystems only subscribe to the primary address of the primary address type.

Package Impact

Filename: rmsmfm_partnerb.pls

Function Level Description – ADDTOQ

```
Function: ADDTOQ
               (O_error_mesage        OUT     VARCHAR2,
                I_message_type        IN      VARCHAR2,
                I_functional_keys     IN      PARTNER_KEY_REC)
```

This public function puts a partner message on PARTNER_MFQUEUE for publishing to the RIB. It is called from both partner trigger and address trigger. The I_functional_keys will contain partner_type, partner_id and optionally, addr_key.

The information from the PARTNER_MFQUEUE and PARTNER_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

This public procedure performs the same tasks as GETNXT except that it only loops for a specific row in the PARTNER_MFQUEUE table. The record on PARTNER_MFQUEUE must match the passed in sequence number (contained in the ROUTING_INFO).

Function Level Description – PROCESS_QUEUE_RECORD (local)

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

Function Level Description – MAKE_CREATE (local)

This private function is used to create the Oracle Object for the initial publication of a business transaction. I_business_object contains the partner header key values (partner type and partner_id). I_rowid is the rowid of the partner_mfqueue row fetched from GETNXT.

Function Level Description – BUILD_HEADER_OBJECT (local)

```
Function: BUILD_HEADER_OBJECT
              (O_error_msg                OUT            VARCHAR2,
               O_rib_partnerdesc_rec      IN OUT NOCOPY  "RIB_PartnerDesc_REC",
               I_business_obj             IN             PARTNER_KEY_REC)
```

This private function accepts partner header key values (partner type and partner ID), builds and returns a header level DESC Oracle Object.

Function Level Description – BUILD_HEADER_OBJECT (local)

This overloaded private function accepts partner header key values (partner type and partner ID), builds and returns a header level REF Oracle Object.

Function Level Description – BUILD_DETAIL_OBJECTS (local)

This private function is responsible for building detail level DESC Oracle Objects. It builds as many detail Oracle Objects as it can given the passed in message type and business object keys (partner type and partner ID).

Function Level Description – BUILD_SINGLE_DETAIL (local)

This private function takes in an address record and builds a detail level Oracle Object. Also it determines if the address is the primary address of the primary address type and set the DESC Oracle Object accordingly.

Function Level Description – BUILD_DETAIL_CHANGE_OBJECTS (local)

This private function builds a DESC Oracle Object to publish to the RIB for detail create and detail update messages (DTL_ADD, DTL_UPD). I_business_obj contains the header level key values (partner type and partner ID).

Function Level Description – BUILD_DETAIL_DELETE_OBJECTS (local)

This private function builds a REF Oracle Object to publish to the RIB for detail delete messages (DTL_DEL). I_business_obj contains the header level key values (partner type and partner ID).

Function Level Description – LOCK_THE_BLOCK (local)

This private function locks all queue records for the current business object (partner type and partner ID). This is to ensure that GETNXT and PUB_RETRY do 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 the queue table for DTL_UPD, DTL_DEL, and HDR_DEL messages.

Function Level Description – HANDLE_ERRORS (local)

This private procedure is called from GETNXT and PUB_RETRY when an exception is raised. I_seq_no is the sequence number of the driving PARTNER_MFQUEUE record. I_function_keys contains detail level key values (partner_type, partner_id, addr_key).

If the error is a non-fatal error, HANDLE_ERRORS passes the sequence number of the driving PARTNER_MFQUEUE record back to the RIB in the ROUTING_INFO. It sends back a status of 'H' - Hospital to the RIB as well. It then updates the status of the queue record to 'H', 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' - Error 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' to 'E'.

Function Level Description – DELETE_QUEUE_REC (local)

This private function will delete the records from PARTNER_MFQUEUE table for the sequence no passed in as input parameter.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
PartnerCre	Partner Create Message	PartnerDesc.xsd
PartnerMod	Partner Modify Message	PartnerDesc.xsd
PartnerDel	Partner Delete Message	PartnerRef.xsd
PartnerDtlCre	Partner Detail Create Message	PartnerDtlDesc.xsd
PartnerDtlMod	Partner Detail Modify Message	PartnerDtlDesc.xsd
PartnerDtlDel	Partner Detail Delete Message	PartnerDtlRef.xsd

Design 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.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
PARTNER_PUB_INFO	Yes	Yes	Yes	Yes
PARTNER_MFQUEUE	Yes	Yes	Yes	Yes
PARTNER	Yes	No	No	No
ADDR	Yes	No	Yes	No
ADD_TYPE_MODULE	Yes	No	No	No
RIB_SETTINGS	Yes	No	No	No

Receiver Unit Adjustment Publication API

Functional Area

Receiver Unit Adjustment

Business Overview

When mistakes are made during the receiving process at the store or warehouse, receiver unit adjustments (RUAs) are made to correct the mistake. RMS publishes messages about receiver unit adjustments to the Oracle Retail Integration Bus (RIB).

When RUAs are initiated through Oracle Retail Invoice Matching (ReIM) or created through RMS forms, a message is published to a store management system (such as SIM) and a warehouse management system. (Note: Oracle Retail's warehouse management system RWMS does NOT subscribe to Receiver Unit Adjustment messages). Because these systems only have access to the original receipt, the message communicates the original receipt number and not the child receipt number.

Package Impact

Business object ID

None

Package name

RMSMFM_RCVUNITADJ

Spec file name: rmsmfm_rcvunitadjs.pls

Body file name: rmsmfm_rcvunitadjb.pls

Package Specification – Global Variables

```
FAMILY            CONSTANT   RIB_SETTINGS.FAMILY%TYPE      'rcvunitadj';
RCVUNITADJ_ADD    CONSTANT   VARCHAR2(15)                  'rcvunitadjcre';
```

Function Level Description – ADDTOQ

```
ADDTOQ (O_error_msg    IN OUT  VARCHAR2,
        I_message_type IN      VARCHAR2,
        I_business_obj IN      RCVUNITADJ_KEY_REC)
```

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 location ID, calculate the thread value.

Insert a record into the RCVUNITADJ_MFQUEUE.

Function Level Description – GETNXT

```
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 driving cursor will query for unpublished records on the RCVUNITADJ_MFQUEUE table (PUB_STATUS = 'U').

GETNXT should check for records on the queue with a status of 'H'ospital for the current business object, GETNXT should raise an exception to send the current message to the Hospital.

The information from the RCVUNITADJ_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

```
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)
```

This procedure republishes the entity that failed to be published before. It is the same as GETNXT except that the record on RCVUNITADJ_MFQUEUE to be published must match the passed in sequence number contained in the ROUTING_INFO.

Function Level Description – PROCESS_QUEUE_RECORD (local)

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.

The function first calls MAKE_CREATE to build the appropriate oracle object. It then calls the DELETE_QUEUE_REC to delete the RUA_MFQUEUE for the passed-in rowid.

Function Level Description – MAKE_CREATE (local)

This function is used to create the Oracle Object for the initial publication of a business transaction.

- Call BUILD_HEADER_OBJECT to get a header level Oracle Object plus any extra functional holders.
- Call BUILD_DETAIL_OBJECTS to get a table of detail level Oracle objects and add the detail level Oracle Objects to the header object.

Function Level Description – BUILD_HEADER_OBJECT (local)

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

This function also builds the routing information object using the location.

Function Level Description – BUILD_DETAIL_OBJECTS (local)

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.

- Call BUILD_SINGLE_DETAIL passing in the I_business_obj record.
- Ensure that ROUTING_INFO is constructed if routing information is stored at the detail level in the business transaction.

Function Level Description – BUILD_SINGLE_DETAIL (local)

Accept inputs and builds a detail level Oracle Object. If the adjustment quantity is negative, the from disposition should be 'ATS' and the to disposition should be NULL. If the adjustment quantity is positive, the to disposition should be NULL and the from disposition should be 'ATS'.

Function Level Description – LOCK_THE_BLOCK (local)

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 the queue table for DTL_UPD, DTL_DEL, and HDR_DEL messages.

Function Level Description – HANDLE_ERRORS (local)

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 RCVUNITADJ_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.

Function Level Description – DELETE_QUEUE_REC (local)

This private function will delete the records from rcvunitadj_mfqueue table for the rowid passed in as input parameter.

Trigger Impact

Trigger name: EC_TABLE_RUA_AIR.TRG

Trigger file name: ec_table_rua_air.trg

Table: RAU_RIB_INTERFACE

Inserts

- Send the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_RCVUNITADJ.RCVUNITADJ_ADD.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
RcvUnitAdjCre	Receiver Unit Adjustment Create Message	RcvUnitAdjDesc.xsd

Design Assumptions

- Each receiver unit adjustment contains the delta quantity to be adjusted. As such they can be processed in any order by the subscribing application. There is no dependency between different RUA messages.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
RUA_MFQUEUE	Yes	Yes	Yes	Yes

RTV Request Publication API

Functional Area

Return to Vendor

Business Overview

A return to vendor (RTV) order is used to send merchandise back to the supplier. The RTV message is published by RMS to the store or warehouse. For an RTV, the initial transfer of stock to the store is a distinctly different step from the RTV itself. Once the transferred stock arrives at the store, the user then creates the RTV. RTVs are created by the following:

1. Adding one supplier.
2. Selecting the sending locations.
3. Adding the items, either individually or through the use of item lists.

In order to return items to a vendor from multiple stores as part of one operation, the items must go through a single warehouse. The transfer of items from several different stores to one warehouse is referred to as a mass return transfer (MRT). The items are subsequently returned to the vendor from the warehouse.

Return to vendor requests created in RMS should be published to the RIB to provide the integration subsystem application with visibility to the corporately created RTV. Consequently, when the integration subsystem application ships the RTV, it must communicate the original RTV order number back to RMS so that RMS can correctly update the original RTV record.

Package Impact

Business Object ID

RTV order number.

Package name: RMSMFM_RTVREQ

Spec file name: rmsmfm_rtvreqs.pls

Body file name: rmsmfm_rtvreqb.pls

Function Level Description – ADDTOQ

```
ADDTOQ (O_error_msg    IN OUT  VARCHAR2,
        I_message_type IN      VARCHAR2,
        I_rtv_order_no IN      RTV_HEAD.RTV_ORDER_NO%TYPE,
        I_status       IN      RTV_HEAD.STATUS_IND%TYPE,
        I_rtv_seq_no   IN      RTV_DETAIL.SEQ_NO%TYPE,
        I_item         IN      RTV_DETAIL.ITEM%TYPE,
        I_publish_ind  IN      RTV_DETAIL.PUBLISH_IND%TYPE)
```

There are some tasks relating to streamlining the queue clean up process that need to occur in ADDTOQ. The goal is to have at most one record on the queue for business transactions up until their initial publication.

- For header level insert messages (HDR_ADD), inserts a record in the RTVREQ_PUB_INFO table. The published flag is set to 'N'. The correct thread for the business transaction is calculated and written. Calls API_LIBRARY.RIB_SETTINGS to get the number of threads used for the publisher. Using the number of threads, and the business object id, calculates the thread value.
- For all records except header level inserts (HDR_ADD), the thread_no, initial_approval_ind, and shipped_ind are queried from the RTVREQ_PUB_INFO table.
- If the business transaction has not been approved (initial_approval_ind = 'N') or it has already been shipped (shipped_ind = 'Y') and the triggering message is one of DTL_ADD, DTL_UPD, DTL_DEL, HDR_DEL, no processing will take place and the function exits.
- For detail level messages deletes (DTL_DEL), the system only needs one (the most recent) record per detail in the RTVREQ_MFQUEUE. Any previous records that exist on the RTVREQ_MFQUEUE for the record that has been passed are deleted. If the publish_ind is 'N', the DTL_DEL message is not added to the queue.
- For detail level message deletes (DTL_UPD), the system only needs one DTL_UPD (the most recent) record per detail in the RTVREQ_MFQUEUE. Any previous DTL_UPD records that exist on the RTVREQ_MFQUEUE for the record that has been passed are deleted. The system does not want to delete any detail inserts that exist on the queue for the detail. The system ensures subscribers are not passed a detail modification message for a detail that they do not yet have.
- For header level delete messages (HDR_DEL), deletes every record in the queue for the business transaction.
- For header level update message (HDR_UPD), updates the RTVREQ_PUB_INFO.INITIAL_APPROVAL_IND to 'Y' if the business transaction is in approved status (status of '10').

- For header level update message (HDR_UPD), updates the RTVREQ_PUB_INFO.SHIPPED_IND to 'Y' if the business transaction is in shipped status (status of '15').
- For all records except header level inserts (HDR_ADD), inserts a record into the RTVREQ_MFQUEUE.

Function Level Description – GETNXT

```
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)
```

LP_error_status is initialized to API_CODES.HOSPITAL at the beginning of GETNXT.

The RIB calls GETNXT to get messages. It performs a cursor loop on the unpublished records on the RTVREQ_MFQUEUE table (PUB_STATUS = 'U'). It only needs 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 RTVREQ_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 executes more than one iteration in 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, it removes the header delete message from the queue and loops again.
2. The queue is locked for the current business object.

The information from the RTVREQ_MFQUEUE and RTVREQ_PUB_INFO table is passed to PROCESS_QUEUE_RECORD. PROCESS_QUEUE_RECORD builds 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

```
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)
```

This procedure republishes the entity that failed to be published before. It is the same as GETNXT except that the record on RTVREQ_MFQUEUE to be published must match the passed in sequence number contained in the ROUTING_INFO.

Function Level Description – PROCESS_QUEUE_RECORD (local)

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.

It checks to see if the business object is being published for the first time. If the published_ind on the PUB_INFO table is 'N' or 'I', the business object is being published for the first time. If so, calls MAKE_CREATE.

Otherwise,

If the record from RTVREQ_MFQUEUE table is a header update (HDR_UPD).

- Calls BUILD_HEADER_OBJECT to build the Oracle Object to publish to the RIB. This will also populate the ROUTING_INFO.
- Updates RTVREQ_PUB_INFO with updated new header information
- Deletes the record from the RTVREQ_MFQUEUE table.

If the record from RTVREQ_MFQUEUE table is a detail add or update (DTL_ADD, DTL_UPD).

- Calls BUILD_HEADER_OBJECT to build the header portion of the Oracle Object to publish to the RIB. This also populates the ROUTING_INFO.
- Calls BUILD_DETAIL_CHANGE_OBJECTS to build the detail portion of the Oracle Object. This also takes care of any RTVREQ_MFQUEUE deletes.

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

- Calls BUILD_HEADER_OBJECT to build the header portion of the Oracle Object to publish to the RIB. This also populates the ROUTING_INFO.
- Calls BUILD_DETAIL_DELETE_OBJECTS to build the detail portion of the Oracle Object. This also takes care of any RTVREQ_MFQUEUE deletes.

Function Level Description – MAKE_CREATE (local)

This function is used to create the Oracle Object for the initial publication of a business transaction.

- Calls BUILD_HEADER_OBJECT to build the Oracle Object to publish to the RIB. This also populates the ROUTING_INFO.
- Calls BUILD_DETAIL_OBJECTS to get a table of detail level Oracle objects and a table of RTVREQ_MFQUEUE rowids to delete.
- Deletes records from the RTVREQ_MFQUEUE for all rowids returned by BUILD_DETAIL_OBJECTS. Deletes are done by rowids instead of business transaction keys to ensure that noting is deleted off the queue that has not been published.
- If the entire business transaction was added to the Oracle Object, also deletes the RTVREQ_MFQUEUE record that was picked up by GETNXT. If the entire business transaction was not published we need to leave something on the RTVREQ_MFQUEUE to ensure that the rest of it is picked up by the next call to GETNXT.
- The header and detail level Oracle Objects are combined and returned.

Function Level Description – BUILD_HEADER_OBJECT (local)

Take all necessary data from RTV_HEAD table and put it into a "RIB_RTVReqDesc_REC" and "RIB_RTVReqRef_REC" object.

Puts the location into the ROUTING_INFO.

Function Level Description – BUILD_DETAIL_CHANGE_OBJECTS (local)

Calls BUILD_DETAIL_OBJECTS.

BUILD_DETAIL_OBJECTS creates a table of RTVREQ_MFQUEUE ROWIDs to delete. Deletes these records.

BUILD_DETAIL_OBJECTS creates a table of RTV_DETAIL ROWIDs to update. Updates the PUBLISH_IND to Y for these records.

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

Function Level Description – BUILD_DETAIL_OBJECTS (local)

The function is responsible for building the detail level Oracle Objects. It builds as many detail Oracle Object as it can given the passed in message type and business object keys.

If the function is being called from MAKE_CREATE:

- Selects any unpublished detail records from the business transaction (RTV_DETAIL.PUBLISH_IND will be 'N'). Creates Oracle Objects for details that are selected by calling BUILD_SINGLE_DETAIL.
- Ensures that the PUBLISH_IND gets set to Y for each RTV_DETAIL record placed into the Oracle Objects. A table of ROWIDs to update is created in BUILD_DETAIL_OBJECTS. The actual update statement occurs in BUILD_DETAIL_CHANGE_OBJECTS using this table of ROWIDS.
- Ensures that RTVREQ_MFQUEUE is deleted from as needed. If there is more than one RTVREQ_MFQUEUE record for a detail level record, makes sure they all get deleted. The system only cares about current state, not every change. A table of ROWIDs to delete is created in BUILD_DETAIL_OBJECTS. The actual delete statement occurs in BUILD_DETAIL_CHANGE_OBJECTS using this table of ROWIDS.
- Ensures that no more than MAX_DETAILS_TO_PUBLISH records are put into Oracle Objects.
- Ensures that the detail records being added to the object have not already been published. This can happen if GETNXT was previously called for the current business object, and the MAX_DETAILS_TO_PUBLISH limit had been reached. The system ensures these details do not get added again by looking at each detail's PUBLISH_IND.

If the function is not being called from MAKE_CREATE:

- Selects any records on the RTVREQ_MFQUEUE that are for the same business object ID. Fetches the records in order of seq_no on the MFQUEUE table.
- Ensures that RTVREQ_MFQUEUE is deleted from as needed. A table of ROWIDs to delete will be created in BUILD_DETAIL_OBJECTS. The actual delete statement occurs in BUILD_DETAIL_CHANGE_OBJECTS using this table of ROWIDS.
- If the message type is a detail create (DTL_ADD), ensures that the PUBLISH_IND gets set to Y for each RTV_DETAIL record placed into the Oracle Objects. A table of ROWIDs to update will be created in BUILD_DETAIL_OBJECTS. The actual update statement occur in BUILD_DETAIL_CHANGE_OBJECTS using this table of ROWIDS.

A concern here is making sure that the system does not delete information from the queue table that has not been published. For this reason, the system performs deletes by ROWID. The system also attempts to get everything in the same cursor to ensure that the message we published matches the deletes we perform from the RTVREQ_MFQUEUE table regardless of trigger execution during GETNXT calls.

Function Level Description – BUILD_DETAIL_DELETE_OBJECTS (local)

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 – BUILD_SINGLE_DETAIL (local)

Puts the inputted information in a RIB_RTVREQDTL_TBL object.

Function Level Description – LOCK_THE_BLOCK (local)

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 the queue table for DTL_UPD, DTL_DEL, and HDR_DEL messages.

Function Level Description – DELETE_QUEUE_REC (local)

Deletes a record from the RTVREQ_MFQUEUE table, using the passed in sequence number.

Function Level Description – HANDLE_ERRORS (local)

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

Trigger name: EC_TABLE_RHD_AIUDR.TRG

Trigger file name: ec_table_rhd_aiudr.trg

Table: RTV_HEAD

A trigger on the RTV_HEAD table captures Inserts, Updates, and Deletes.

- **Inserts:** Sends the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_RTVREQ.HDR_ADD.
- **Updates:** Sends the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_RTVREQ.HDR_UPD.
- **Deletes:** Sends the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_RTVREQ.HDR_DEL.

Trigger name: EC_TABLE_RDT_AIUDR.TRG

Trigger file name: ec_table_rdt_aiudr.trg

Table: RTV_DETAIL

A trigger on the RTV_DETAIL table captures Inserts, Updates, and Deletes.

- **Inserts:** Sends the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_RTVREQ.DTL_ADD.
- **Updates:** Sends the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_RTVREQ.DTL_UPD.
- **Deletes:** Sends the appropriate column values to the ADDTOQ procedure in the MFM with the message type RMSMFM_RTVREQ.DTL_DEL.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
RtvReqCre	RTV Request Create Message	RTVReqDesc.xsd
RtvReqMod	RTV Request Modify Message	RTVReqDesc.xsd
RtvReqDel	RTV Request Delete Message	RTVReqRef.xsd
RtvReqDtlCre	RTV Request Detail Create Message	RTVReqDesc.xsd
RtvReqDtlMod	RTV Request Detail Modify Message	RTVReqDesc.xsd
RtvReqDtlDel	RTV Request Detail Delete Message	RTVReqRef.xsd

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 remove the lock. It is assumed that this time will be fairly short (at most 2-3 seconds). It is also assumed that this will occur rarely because 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.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
RTVREQ_MFQUEUE	Yes	Yes	Yes	Yes
RTVREQ_PUB_INFO	Yes	Yes	Yes	Yes
RTV_HEAD	Yes	No	No	No
RTV_DETAIL	Yes	No	No	No

Seed Data Publication API

Functional Area

Foundation Data

Business Overview

Seed data publication to the RIB allows RMS to send some basic foundation data information to external systems to seed their database. The data contained in this API is usually fairly static and does not frequently change after initial implementation.

Some examples of seed data include diff types, item types, carriers, shipping methods, supplier types, location types, order types and return reasons.

Package Impact

File name: rmsmfm_seeddatas/b.pls

Function Level Description – ADDTOQ

```
PROCEDURE: ADDTOQ
               (O_status      OUT            VARCHAR2,
                O_text        OUT            VARCHAR2,
I_message_type IN     CODES_MFQUEUE.MESSAGE_TYPE%TYPE,
            I_code_type       IN             CODES_MFQUEUE.CODE_TYPE%TYPE,
            I_message         IN OUT         rib_sxw.SXWHandle)
```

This procedure is called by the table triggers EC_TABLE_CODEHD_AIUDR, EC_TABLE_CODEDTL_AIUDR and EC_TABLE_DIFF_TYPE_AIUDR. The procedure accepts a message variable that consists of the code or diff information in XML tags, a code type variable (this will be hard coded 'OOOO' for diff types) and one of the message types defined in the package specification. It inserts a row into the message family queue CODES_MFQUEUE along with the passed in values and the next sequence number from the message family sequence, and sets the status to unpublished. The procedure will then call API_LIBRARY.WRITE_DOCUMENT_STR which will return 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            CODES_MFQUEUE.MESSAGE_TYPE%TYPE,
            O_message         OUT            nocopy CLOB,
            O_code_type       OUT            CODES_MFQUEUE.CODE_TYPE%TYPE)
```

This publicly exposed procedure is called by the RIB publication adaptor. The message type is the RIB defined short message name.

The error text parameter contains application-generated information, such as the application's sequence number of the message that failed, and the Oracle or other error that occurred when the retrieval failed.

This procedure will call the internal function DO_GETNXT which will actually retrieve the clob from the CODES_MFQUEUE table so that it may be published to the RIB.

Function Level Description – DO_GETNXT (local)

This internal procedure will select the record from the CODES_MFQUEUE table having the lowest sequence number and a pub_status of 'U'. It will return the clob, the message type and code type to the out parameters to be passed back to GETNXT. The procedure will then call the DELETE_QUEUE_REC function to delete the record that is being published.

Function Level Description – DELETE_QUEUE_REC (local)

This procedure will delete the record from the CODES_MFQUEUE table that has the sequence number corresponding to the I_seq_no parameter.

File name: code_head_xmls/b.pls

Function Level Description – BUILD_MESSAGE

If the I_action_type is 'D' (a record is being deleted), an internal variable holding the doc type will be set to RMSMFM_SEEDDATA.HDR_REF_MSG. The function will then call API_LIBRARY.CREATE_MESSAGE_STR to return a clob that has the appropriate structure for the code head delete message. The function then calls the DELETE_CODE_HEAD function to populate the clob that was created.

If the I_action_type is not 'D' (a record has been added or updated), an internal variable holding the doc type will be set to RMSMFM_SEEDDATA.HDR_DESC_MSG. The function will then call API_LIBRARY.CREATE_MESSAGE_STR to return a clob that has the appropriate structure for the code head desc message. The function then calls the ADD_UPDATE_CODE_HEAD function to populate the clob that was created.

Function Level Description – DELETE_CODE_HEAD

This function will accept a record that holds code_head values. The rib_sxw.addElement function will be called to add the code type from the I_code_head_rec to the clob (or root).

Function Level Description – ADD_UPDATE_CODE_HEAD

This function will accept a record that holds CODE_HEAD values. The rib_sxw.addElement function will be called to add the code type and code type description from the I_code_head_rec to the clob (or root).

File name: code_detail_xmls/b.pls

Function Level Description – BUILD_MESSAGE

If the I_action_type is 'D' (a record is being deleted), an internal variable holding the doc type should be set to RMSMFM_SEEDDATA.DTL_REF_MSG. The function will then call API_LIBRARY.CREATE_MESSAGE_STR to return a clob that has the appropriate structure for the code detail delete message. The function then calls the DELETE_CODE_DETAIL function to populate the clob that was created.

If the I_action_type is not 'D' (a record has been added or updated), an internal variable holding the doc type should be set to RMSMFM_SEEDDATA.DTL_DESC_MSG. The function will then call API_LIBRARY.CREATE_MESSAGE_STR to return a clob that has the appropriate structure for the code detail desc message. The function then calls the ADD_UPDATE_CODE_DETAIL function to populate the clob that was created.

Function Level Description – DELETE_CODE_DETAIL

```
FUNCTION DELETE_CODE_DETAIL
               (O_status             OUT     VARCHAR2,
                O_text               OUT     VARCHAR2,
                I_code_detail_rec    IN      CODE_DETAIL%ROWTYPE,
                Root                 IN      OUT rib_sxw.SXWHandle)
```

This function will accept a record that holds code_detail values. The rib_sxw.addElement function will be called to add the code type and code from the I_code_detail_rec to the clob (or root).

Function Level Description – ADD_UPDATE_CODE_DETAIL

This function will accept a record that holds code_detail values. The rib_sxw.addElement function will be called to add the code type, code, code description, required indicator and code sequence from the I_code_detail_rec to the clob (or root).

File name: diff_type_xmls/b.pls

Function Level Description – BUILD_MESSAGE

If the I_action_type is 'D' (a record is being deleted), an internal variable holding the doc type will be set to RMSMFM_SEEDDATA.DIFF_TYPE_REF_MSG. The function will then call API_LIBRARY.CREATE_MESSAGE_STR to return a clob that has the appropriate structure for the diff type delete message. The function then calls the DELETE_DIFF_TYPE function to populate the clob that was created.

If the I_action_type is not 'D' (a record has been added or updated), an internal variable holding the doc type will be set to RMSMFM_SEEDDATA.DIFF_TYPE_DESC_MSG. The function will then call API_LIBRARY.CREATE_MESSAGE_STR to return a clob that has the appropriate structure for the diff type desc message. The function then calls the ADD_UPDATE_DIFF_TYPE function to populate the clob that was created.

Function Level Description – DELETE_DIFF_TYPE

```
FUNCTION DELETE_DIFF_TYPE
               (O_status             OUT     VARCHAR2,
                O_text               OUT     VARCHAR2,
                I_diff_type_rec      IN      DIFF_TYPE%ROWTYPE,
                Root                 IN      OUT rib_sxw.SXWHandle)
```

This function will accept a record that holds diff_type values. The rib_sxw.addElement function will be called to add the diff type from the I_diff_type_rec to the clob (or root).

Function Level Description – ADD_UPDATE_CODE_DETAIL

```
FUNCTION ADD_UPDATE_DIFF_TYPE
               (O_status             OUT           VARCHAR2,
                O_text               OUT           VARCHAR2,
                I_diff_type_rec      IN            DIFF_TYPE%ROWTYPE,
                Root                 IN OUT        rib_sxw.SXWHandle)
```

This function will accept a record that holds diff_type values. The rib_sxw.addElement function will be called to add the diff type and diff type description from the I_diff_type_rec to the clob (or root).

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
CodeHdrCre	Code Head Create Message	CodeHdrDesc.xsd
CodeHdrMod	Code Head Modify Message	CodeHdrDesc.xsd
CodeHdrDel	Code Head Delete Message	CodeHdrRef.xsd
CodeDtlCre	Code Detail Create Message	CodeDtlDesc.xsd
CodeDtlMod	Code Detail Modify Message	CodeDtlDesc.xsd
CodeDtlDel	Code Detail Delete Message	CodeDtlRef.xsd
DiffTypeCre	Diff Type Create Message	DiffTypeDesc.xsd
DiffTypeMod	Diff Type Modify Message	DiffTypeDesc.xsd
DiffTypeDel	Diff Type Delete Message	DiffTypeRef.xsd

Design Assumptions

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
CODES_MFQUEUE	Yes	Yes	No	Yes
CODE_HEAD	Yes	Yes	No	Yes
CODE_DETAIL	Yes	Yes	No	Yes

Seed Object Publication API

Functional Area

Foundation Data

Business Overview

Seed object publication to the RIB allows RMS to send country information as well as currency rates so that external systems will have all of the latest information regarding countries and currency rates.

Seed object publication consists of a message containing country and currency rate information from the tables COUNTRY and CURRENCY_RATES. One message will be synchronously created and placed in the message queue each time a COUNTRY and CURRENCY_RATES record is created, modified or deleted in RMS. When a COUNTRY or CURRENCY_RATES record is created or modified, the message will contain a full snapshot of the modified record. When a COUNTRY record is deleted, the message will contain a partial snapshot of the deleted record. Messages are retrieved from the message queue in the order they were created.

Package Impact

File name: rmsmfm_seedobjs/b.pls

Function Level Description – ADDTOQ

```
PROCEDURE: ADDTOQ
           (O_error_message   IN OUT         VARCHAR2,
            I_message_type    IN             SEEDOBJ_MFQUEUE.MESSAGE_TYPE%TYPE,
            I_country_id      IN             SEEDOBJ_MFQUEUE.COUNTRY_ID%TYPE,
            I_currency_code   IN             SEEDOBJ_MFQUEUE.CURRENCY_CODE%TYPE,
            I_country_desc    IN             SEEDOBJ_MFQUEUE.COUNTRY_DESC%TYPE,
            I_effective_date  IN             SEEDOBJ_MFQUEUE.EFFECTIVE_DATE%TYPE,
            I_exchange_type   IN             SEEDOBJ_MFQUEUE.EXCHANGE_TYPE%TYPE,
            I_exchange_rate   IN             SEEDOBJ_MFQUEUE.EXCHANGE_RATE%TYPE)
RETURN BOOLEAN;
```

This function is called by either the COUNTRY or CURRENCY_RATES row trigger, and takes the message type and the table values (country_id for COUNTRY table and currency_code for CURRENCY_RATES table). It inserts a row into the message family queue along with the passed in values and the next sequence number from the message family sequence. The pub status will always be 'U' except for create messages, then it will be 'N'. It returns error codes and strings according to the standards of the application in which it is being implemented.

Function Level Description – GETNXT

```
PROCEDURE GETNXT
           (O_status_code     IN OUT         VARCHAR2,
            O_error_msg       IN OUT         RTK_ERRORS.RTK_TEXT%TYPE,
            O_message_type    IN OUT         VARCHAR2,
            O_message         IN OUT         RIB_OBJECT,
            O_bus_obj_id      IN OUT         RIB_BUSOBJID_TBL,
            O_routing_info    IN 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 SEEDOBJ_MFQUEUE table (PUB_STATUS = 'U'). It will only execute one loop iteration in most cases. For each record retrieved, GETNXT checks for records on the queue with a status of 'H' - Hospital. If there are any such records for the current business object, GETNXT raises an exception to send the current message to the Hospital.

The information from the SEEDOBJ_MFQUEUE and 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 SEEDOBJ_MFQUEUE table. The record on SEEDOBJ_MFQUEUE must match the passed in sequence number (contained in the ROUTING_INFO).

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
countrycre	Code Head Create Message	CountryDesc.xsd
countrymod	Code Head Modify Message	CountryDesc.xsd
countrydel	Code Head Delete Message	CountryRef.xsd
curratecre	Code Detail Create Message	CurrRateDesc.xsd
curratemod	Code Detail Modify Message	CurrRateDesc.xsd

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
SEEDOBJ _MFQUEUE	Yes	Yes	No	Yes
COUNTRY	Yes	Yes	Yes	Yes
CURRENCY_RATES	Yes	Yes	Yes	No

Store Publication API

Functional Area

Foundation Data

Business Overview

RMS publishes data about stores in messages to the Oracle Retail Integration Bus (RIB) for other applications that needs to keep their locations synchronized with RMS. RMS publishes messages to the RIB to create, modify, and delete store events for all store types. These messages are triggered by insert/update/delete on the RMS STORE table and/or the ADDR table with module 'ST' (for store). The system only publishes the current state of the store, not every change.

Only the primary address and primary address type are published through this message, as it is assumed that integration subsystems only require one address.

Package Impact

File name: rmsmfm_stores/b.pls

Function Level Description – ADDTOQ

```
Function: ADDTOQ
              (O_error_msg                OUT           VARCHAR2,
               I_message_type             IN            VARCHAR2,
               I_store_key_rec            IN            STORE_KEY_REC,
               I_addr_publish_ind         IN            ADDR.PUBLISH_IND%TYPE)
```

This public function puts a store message on STORE_MFQUEUE for publishing to the RIB. It is called from both store trigger and address trigger. The I_functional_keys will contain store and, optionally, addr_key.

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)
```

This public procedure is called from the RIB to get the next messages. It performs a cursor loop on the unpublished records on the STORE_MFQUEUE table (PUB_STATUS = 'U').

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            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)
```

This public procedure performs the same tasks as GETNXT except that it only loops for a specific row in the STORE_MFQUEUE table. The record on STORE_MFQUEUE must match the passed in sequence number (contained in the ROUTING_INFO).

Function Level Description – PROCESS_QUEUE_RECORD (local)

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

Function Level Description – MAKE_CREATE (local)

This private function is used to create the Oracle Object for the initial publication of a business transaction. I_business_object contains the store header key values (store). I_rowid is the rowid of the store_mfqueue row fetched from GETNXT.

Function Level Description – BUILD_HEADER_OBJECT (local)

This private function accepts store header key value (store), builds and returns a header level DESC Oracle Object.

This overloaded private function accepts store header key value (store), builds and returns a header level REF Oracle Object.

Function Level Description – BUILD_DETAIL_OBJECTS (local)

The private function is responsible for building detail level DESC Oracle Objects. It builds as many detail Oracle Object as it can given the passed in message type and business object keys (store).

Function Level Description – BUILD_SINGLE_DETAIL (local)

This private function takes in an address record and builds a detail level Oracle Object. Also find out if the address is the primary address of the primary address type and set the DESC Oracle Object accordingly.

Function Level Description – BUILD_DETAIL_CHANGE_OBJECTS (local)

This private function builds a DESC Oracle Object to publish to the RIB for detail create and detail update messages (DTL_ADD, DTL_UPD). I_business_obj contains the header level key values (store).

Function Level Description – BUILD_DETAIL_DELETE_OBJECTS (local)

This private function builds a REF Oracle Object to publish to the RIB for detail delete messages (DTL_DEL). I_business_obj contains the header level key values (store).

Function Level Description – LOCK_THE_BLOCK (local)

This private function locks all queue records for the current business object (store). This is to ensure that GETNXT and PUB_RETRY do 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 the queue table for DTL_UPD, DTL_DEL, and HDR_DEL messages.

Function Level Description – HANDLE_ERRORS (local)

This private procedure is called from GETNXT and PUB_RETRY when an exception is raised. I_seq_no is the sequence number of the driving STORE_MFQUEUE record. I_function_keys contains detail level key values (store, addr_key).

If the error is a non-fatal error, HANDLE_ERRORS passes the sequence number of the driving STORE_MFQUEUE record back to the RIB in the ROUTING_INFO. It sends back a status of 'H' - Hospital to the RIB as well. It then updates the status of the queue record to 'H', 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' - Error 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' to 'E'.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
StoreCre	Store Create Message	StoreDesc.xsd
StoreMod	Store Modify Message	StoreDesc.xsd
StoreDel	Store Delete Message	StoreRef.xsd

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
STORE_PUB_INFO	Yes	Yes	Yes	Yes
ADDR	Yes	No	Yes	No
STORE_MFQUEUE	Yes	Yes	Yes	Yes
ADD_TYPE_MODULE	Yes	No	No	No
STORE	Yes	No	No	No

Design 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.

Transfers Publication API

Functional Area

Transfer

Business Overview

A transfer is a movement of stock on hand from one stockholding location within the company to another.

The transfer publication processing publishes transfers in 'Approved' status.

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

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

For a customer order transfer (tsf_type = 'CO'), customer related information is pulled from ORDCUST table. Additional trigger is put on ORDCUST to capture delivery and billing change for the customer order transfer through the transfer message family.

Package Impact

Business Object ID

Transfer number

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

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 unless the transfer is set back to Input status.
3. **Messages**: No messages are created on the queue until the transfer is approved.

Modify Details

1. **Prerequisites**: Only modifications to transfer quantities are 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 has not 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 has not 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) is 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

```
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, the 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), inserts a record in the TRANSFERS_PUB_INFO table. The published flag is set to 'N'. The correct thread for the Business transaction is calculated and written. The functionAPI_LIBRARY.RIB_SETTINGS is called to get the number of threads used for the publisher. Using the number of threads, and the Business object ID, the thread value is calculated.
- For all records except header level inserts (HDR_ADD), the thread_no and initial_approval_ind are 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 will take place and the function exits.
- For detail level message deletes (DTL_DEL), only the most recent record per detail in the TSF_MFQUEUE is required. Any previous records that exist on the TSF_MFQUEUE for the record that has been passed are deleted. If the publish_ind is 'N', the DTL_DEL message is not added to the queue.
- For detail level message updates (DTL_UPD), only the most recent DTL_UPD record per detail in the TSF_MFQUEUE is required. Any previous DTL_UPD records that exist on the TSF_MFQUEUE for the record that has been passed are deleted. The system does not want to delete any detail inserts that exist on the queue for the detail. It ensures subscribers have not passed a detail modification message for a detail that they do not yet have.
- For header level delete messages (HDR_DEL), deletes every record in the queue for the Business transaction.

- For header level update message (HDR_UPD), updates 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), inserts 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

```
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 only needs 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 executes 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, it removes 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, it removes 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 builds 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

```
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)
```

This procedure republishes the entity that failed to be published before. It is the same as GETNXT except that the record on TSF_MFQUEUE to be published must match the passed in sequence number contained in the ROUTING_INFO.

Function Level Description – PROCESS_QUEUE_RECORD (local)

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:

- Calls DELETE_QUEUE_REC to delete the record from TSF_MFQUEUE.

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

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

If the message type is HDR_UNAPRV":

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

If the message type is HDR_ADD or DTL_ADD:

- Calls MAKE_CREATE to publish the entire transfer.

If the record from TSF_MFQUEUE table is HDR_UPD:

- Calls BUILD_HEADER_OBJECT to build the Oracle Object to publish to the RIB and deletes from the queue.

If the record from TSF_MFQUEUE table is DTL_ADD or DTL_UPD:

- Calls 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):

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

This function puts the following in the routing info (RIB_ROUTING_INFO_TBL):

- 'from_phys_loc' – transfer from location. In case of warehouse, it's the physical warehouse.
- 'from_phys_loc_type' - transfer from location type – 'S' for store, 'W' for warehouse, 'E' for external finisher.
- 'to_phys_loc' – transfer to location. In case of warehouse, it's the physical warehouse.
- 'to_phys_loc_type' – transfer to location type. In case of store, 'S' for physical store (i.e. stockholding company store), 'V' for virtual store (i.e. non-stockholding company store).

Function Level Description – MAKE_CREATE (local)

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 RIB. 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 is ready to be published for the first time to the external system(s)).

Function Level Description – BUILD_HEADER_OBJECT (local)

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

Function Level Description – BUILD_DETAIL_OBJECTS (local)

This function is responsible for fetching the detail info and ticket type to be sent to RWMS. 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)

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

Function Level Description – GET_RETAIL (local)

Gets the price and selling unit of measure (UOM) of the item.

Function Level Description – GET_GLOBALS (local)

Get all the system options and variables needed for processing.

Function Level Description – BUILD_DETAIL_CHANGE_OBJECTS (local)

Calls BUILD_DETAIL_OBJECT to publish the record. Updates TSFDETAIL.publish_ind to 'Y' and deletes the record from TSF_MFQUEUE.

Function Level Description – BUILD_DETAIL_DELETE_OBJECTS (local)

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

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

Deletes from TSF_MFQUEUE when done.

Function Level Description – LOCK_THE_BLOCK (local)

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 the queue table for DTL_UPD, DTL_DEL, and HDR_DEL messages.

Function Level Description – LOCK_DETAILS (local)

Locks the transfer details before updating the publish_ind on TSFDETAIL.

Function Level Description – DELETE_QUEUE_REC (local)

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

Function Level Description – HANDLE_ERRORS (local)

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

A trigger on the TSFHEAD and TSFDETAIL exists to capture Inserts, Updates, and Deletes.

Trigger name: EC_TABLE_THD_AIUDR.TRG

Trigger file name: ec_table_thd_aiudr.trg

Table: TSFHEAD

- **Inserts:** Sends the tsf_no and tsf_type level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_Transfers.HDR_ADD.
- **Updates:** Sends the tsf_no and tsf_type level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_Transfers.HDR_UPD.
- **Deletes:** Sends 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:** Sends the tsf_no and item level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_Transfers.DTL_ADD.
- **Updates:**Sends the tsf_no and item level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_Transfers.DTL_UPD.
- **Deletes:** Sends the tsf_no and item level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_Transfers.DTL_DEL.

Trigger name: EC_TABLE_ORC_AUR.TRG

Trigger file name: ec_table_orc_aur.trg

Table: ORDCUST

- **Updates:** For ORDCUST associated with a published 'CO' transfer, send the tsf_no and tsf_type level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_Transfers.HDR_UPD.

Message XSD

Here are the filenames that correspond with each message type. See *Oracle Retail Integration Bus documentation* for each message type in order to get a detailed picture of the composition of each message.

Message Types	Message Type Description	XML Schema Definition (XSD)
TransferCre	Transfer Create Message	TsfDesc.xsd
TransferHdrMod	Transfer Modify Message	TsfDesc.xsd
TransferDel	Transfer Delete Message	TsfRef.xsd
TransferDtlCre	Transfer Detail Create Message	TsfDesc.xsd
TransferDtlMod	Transfer Detail Modify Message	TsfDesc.xsd
TransferDtlDel	Transfer Detail Delete Message	TsfRef.xsd

Design Assumptions

- After a transfer has been approved, Oracle Retail assumes 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 set up to call stored procedures, not stored functions. Any public program then needs to be a procedure.

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
ORDCUST_DETAIL	Yes	No	No	No
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

UDA Publication API

Functional Area

Foundation Data

Business Overview

RMS publishes messages about user-defined attributes (UDAs) to the Oracle Retail Integration Bus (RIB). UDA provides a method for defining attributes and associating the attributes with specific items, items on an item list, or items in a specific department, class, or subclass. UDAs are useful for information and reporting purposes. Unlike traits or indicators, UDAs are not interfaced with external systems. UDAs do not have any programming logic associated with them. UDA messages are specific to basic UDA identifiers and values defined in RMS. The UDAs can be displayed in one or more of three formats: Dates, Freeform Text, or a List of Values (LOV).

The created messages in the XML builder adds the messages to the UDA_MFQUEUE table which must be published in the same order as they occur in the RMS database.

Package Impact

File name: rmsmfm_udas/b.pls

Function Level Description – ADDTOQ

```
Function: ADDTOQ
              (O_status             OUT     VARCHAR2,
               O_text               OUT     VARCHAR2,
               I_message_type       IN      UDA_MFQUEUE.MESSAGE_TYPE%TYPE,
               I_uda_id             IN      UDA.UDA_ID%TYPE,
               I_uda_value          IN      UDA_VALUES.UDA_VALUE%TYPE,
               I_display_taype      IN      UDA_MFQUEUE.DISPLAY_TYPE%TYPE,
               I_message            IN      CLOB
                          )
```

This procedure is called by the triggers EC_TABLE_UDA_AIUDR and EC_TABLE_UDV_AIUDR and takes the message type, uda_id and uda_value if there is one and the message itself. It inserts a row into the UDA_MFQUEUE along with the passed in values and the next sequence number from the UDA_MFSEQUENCE, setting the status to 'U'npublished. It returns error codes and strings.

Function Level Description – GETNXT

```
Procedure: GETNXT
              (O_status_code        OUT     VARCHAR2,
               O_error_msg          OUT     VARCHAR2,
               O_message_type       OUT     UDA_MFQUEUE.MESSAGE_TYPE%TYPE,
               O_message            OUT     CLOB,
               O_uda_id             OUT     UDA.UDA_ID%TYPE,
               O_uda_value          OUT     UDA_VALUES.UDA_VALUE%TYPE,
               O_display_type       OUT     UDA_MFQUEUE.DISPLAY_TYPE%TYPE
              )
```

This publicly exposed procedure is typically called by a RIB publication adaptor. This procedure's parameters are well defined and arranged in a specific order. The message type is the RIB defined short message name; the message is the XML message; and the uda_id and uda_value are the keys for the message as pertains to the UDA family, not all of which will necessarily be populated for all message types. The status code is one of five values.

Message XSD

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	XML Schema Definition (XSD)
UDAHdrCre	UDA Create Message	UDADesc.xsd
UDAHdrMod	UDA Modify Message	UDADesc.xsd
UDAHdrDel	UDA Delete Message	UDARef.xsd
UDAValCre	UDA_Values Create Message	UDAValDesc.xsd
UDAValMod	UDA_Values Modify Message	UDAValDesc.xsd
UDAValDel	UDA_Values Delete Message	UDAValRef.xsd

Design Assumptions

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
UDA_MFQUEUE	Yes	Yes	No	No
UDA	Yes	Yes	Yes	Yes
UDA_VALUES	Yes	Yes	Yes	Yes

Vendor Publication API

Functional Area

Foundation Data

Business Overview

RMS publishes supplier and supplier address information to the RIB for RWMS and other integration subsystems. Supplier information is published when new suppliers are created, updates are made to existing suppliers or existing suppliers are deleted. Similarly, addresses are published when they are added, modified or deleted. The address types that are published as part of this message are Returns (3), Order (4), and Invoice (5).

As suppliers and addresses are added in RMS, an event capture trigger creates a message that is added to the SUPPLIER_MFQUEUE table.

Package Impact

File name: rmsmfm_vendors/b.pls

Function Level Description – ADDTOQ

```
Function: ADDTOQ
              (I_message_type       IN     VARCHAR2,
               I_supplier           IN     sups.supplier%TYPE,
               I_addr_seq_no        IN     addr.seq_no%TYPE,
               I_addr_type          IN     addr.addr_type%TYPE,
               I_ret_allow_ind      IN     VARCHAR2,
               I_invc_match_ind     IN     VARCHAR2,
               I_org_unit           IN     VARCHAR2,
               I_message            IN     CLOB,
               O_status             OUT    VARCHAR2,
               O_text               OUT    VARCHAR2)
```

This procedure is called by the triggers, and takes the message type, supplier, addr_seq_no, addr_type, ret_allow_ind, and invc_match_ind values, and org_unit and, the message itself. It inserts a row into the supplier message family queue along with the passed in values and the next sequence number from the supplier message family sequence, setting the status to unpublished. It returns error codes and strings.

Function Level Description – GETNXT

```
Procedure: GETNXT
              (O_status_code        OUT    VARCHAR2,
               O_error_msg          OUT    VARCHAR2,
               O_message_type       OUT    VARCHAR2,
               O_message            OUT    CLOB,
               O_supplier           OUT    sups.supplier%TYPE
               O_addr_seq_no        OUT    addr.seq_no%TYPE
               O_addr_type          OUT    addr.addr_type%TYPE
```

This publicly exposed procedure is called by a RIB publication adaptor. The message type is the RIB defined short message name, the message is the xml message, and the family key(s) are the key for the message as pertains to the family, not all of which will necessarily be populated for all message types. The keys for supplier are supplier, addr_seq_no, and addr_type.

Function Level Description – CREATE_PREVIOUS (local)

This procedure determines if a supplier create already exists on the queue table for the same supplier and with a sequence number less than the current records sequence number.

Function Level Description – CLEAN_QUEUE (local)

This procedure cleans up the queue by eliminating modification messages. It is only called if CREATE_PREVIOUS returns true. For each address modification message type, it finds the previous address create message type. It then calls REPLACE_QUE_ADR to copy the modify message into the create message and calls DELETE_QUEUE_REC to delete the modify record. For each delete message type, it finds the previous corresponding create message type. It then calls DELETE_QUEUE_REC to delete the create message record. For each supplier modification message type, it finds the previous supplier create message type. It then calls REPLACE_QUE_SUP to copy the modify message into the create message and calls DELETE_QUEUE_REC to delete the modify record.

Function Level Description – CAN_CREATE (local)

This procedure determines if a complete hierarchical supplier message can be created from the current address and prior address messages in the queue for the same supplier. It checks to see if there is a type 3, 4, or 5 address already in the queue. If the ret_allow_ind is 'Y' and there is a type 3 address, then a ret_flag is set to true. If the invc_match_ind is 'Y' and there is a type 5 address, then a invc_flag is set to true. If all the flags are true, then it returns true because the complete hierarchical message can be created.

Function Level Description – MAKE_CREATE (local)

This procedure combines the current message and all previous messages with the same supplier in the queue table to create the complete hierarchical message. It first creates a new message with the VendorDesc document type. It then gets the supplier create message and adds it to the new message. The remainder of this procedure gets each of the addresses 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.

Function Level Description – DELETE_QUEUE_REC (local)

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

Function Level Description – REPLACE_QUEUE_SUP (local)

This procedure replaces the message in the create supplier record with the message from the modify supplier record.

Function Level Description – REPLACE_QUEUE_ADR (local)

This procedure replaces the message in the create address record with the message from the modify address record.

Function Level Description – CHECK_STATUS (local)

This procedure raises an exception if the status code is set to Error. This will be called immediately after calling a procedure that sets the status code. Any procedure that calls CHECK_STATUS must have its own exception handling section.

Function Level Description – MAKE_CREATE_POU (local)

This procedure is called when message type is 'VendorOUCre' or 'VendorOUDel'. It first creates a new message with the VendorDesc document type. It then gets the Vendor OrgUnit create message and adds it to the new message. This new message is passed back to the bus.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
VendorCre	Vendor Create	VendorDesc.xsd

Message Types	Message Type Description	XML Schema Definition (XSD)
VendorHdrMod	Vendor Header Modify	VendorHdrDesc. xsd
VendorDel	VendorDelete	VendorRef.xsd
VendorAddrCre	Vendor Address Create	VendorAddrDesc.xsd
VendorAddrMod	Vendor Address Modify	VendorAddrDesc.xsd
VendorAddrDel	Vendor Address Delete	VendorAddrRef.xsd
VendorOUCre	Vendor OrgUnit Create	VendorOUDesc.xsd
VendorOUDel	Vendor OrgUnit Delete	VendorOURef.xsd

Design Assumptions

- The adaptor is only setup to call stored procedures, not stored functions. Any public program then needs to be a procedure.
- Once all criteria are met for a valid create message, the messages will be combined and sent to the RIB.
- Messages for supplier and address modifications and deletions will be sent as they are created. An address modification can be sent without the supplier information.
- When multiple set of books is enabled in RMS, org units are required elements when creating a supplier. Addition and deletes from this table are sent either as standalone message or part of the supplier create message.
- When Supplier Sites functionality is enabled, only supplier site data is published. The Supplier level data are not published.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
SUPS	Yes	No	No	No
ADDR	Yes	No	No	No
SUPPLIER_MFQUEUE	Yes	Yes	Yes	Yes
DUAL	Yes	No	No	No
PARTNER_ORG_UNIT	Yes	No	No	No

Warehouse Publication API

Functional Area

Foundation Data

Business Overview

RMS publishes data about warehouses in messages to the Oracle Retail Integration Bus (RIB). Other applications that need to keep their locations synchronized with RMS subscribe to these messages. RMS publishes information about all the warehouses, including both physical and virtual. Those applications on the RIB that understands virtual locations can subscribe to all warehouse messages that RMS publishes. Those applications that do not have virtual location logic, such as SIM and RWMS, it depends on RIB to transform RMS warehouse messages for physical warehouses only.

These RIB messages are triggered on inserting, updating, and deleting of warehouse and warehouse address in the RMS WH table, and the ADDR table with the module 'WH'. Only the primary address of the primary address type is included in this message. Oracle Retail publishes only the current state of the warehouse, not every change.

Package Impact

File name: rmsmfm_whs/b.pls

Function Level Description – ADDTOQ

```
Function: ADDTOQ
           (O_error_mesage           OUT     VARCHAR2,
            I_message_type           IN      VARCHAR2,
            I_wh_key_rec             IN      WH_KEY_REC,
            I_addr_publish_ind       IN      ADDR.PUBLISH_IND%TYPE)
```

This public function puts a warehouse message on WH_MFQUEUE for publishing to the RIB. It is called from both wh trigger and address trigger. The I_functional_keys contains wh and, optionally, addr_key.

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);
```

This public procedure is called from the RIB to get the next messages. It performs a cursor loop on the unpublished records on the WH_MFQUEUE table (PUB_STATUS = 'U'). 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

This public procedure performs the same tasks as GETNXT except that it only loops for a specific row in the WH_MFQUEUE table. The record on WH_MFQUEUE must match the passed in sequence number (contained in the ROUTING_INFO).

Function Level Description – PROCESS_QUEUE_RECORD (local)

This private function controls the building of Oracle Objects (DESC or REF) given the business transaction's key values and a message type. It contains all of the shared processing between GETNXT and PUB_RETRY. (Note: message_type of HDR_ADD can potentially be changed to a DTL_ADD in PROCESS_QUEUE_RECORD).

Function Level Description – DELETE_QUEUE_REC (local)

This private function deletes a record in WH_MFQUEUE table given the row ID.

Function Level Description – MAKE_CREATE (local)

```
Procedure: MAKE_CREATE
              (O_error_msg          OUT           VARCHAR2,
               O_message            IN  OUT       nocopy RIB_OBJECT,
               O_routing_info       IN  OUT       nocopy RIB_ROUTINGINFO_TBL,
               I_wh_key_rec         IN            WH_KEY_REC,
               I_rowid              IN            ROWID)
```

This private function is used to create the Oracle Object for the initial publication of a business transaction. I_business_object contains the warehouse header key values (wh). I_rowid is the rowid of the wh_mfqueue row fetched from GETNXT.

Function Level Description – BUILD_HEADER_OBJECT (local)

```
Procedure: BUILD_HEADER_OBJECT
              (O_error_msg          OUT           VARCHAR2,
               O_routing_info       IN  OUT       nocopy RIB_ROUTINGINFO_TBL,
               O_rib_whdesc_rec     OUT           RIB_WH_DESC,
               I_wh_key_rec         IN            WH_KEY_REC)
```

This private function accepts warehouse header key values (wh), builds and returns a header level DESC Oracle Object.

Function Level Description – BUILD_HEADER_OBJECT (local)

This overloaded private function accepts warehouse header key value (wh), builds and returns a header level REF Oracle Object.

Function Level Description – BUILD_DETAIL_OBJECTS (local)

The private function is responsible for building detail level DESC Oracle Objects. It builds as many detail Oracle Object as it can given the passed in message type and business object keys (wh).

Function Level Description – BUILD_SINGLE_DETAIL (local)

This private function takes in an address record and builds a detail level Oracle Object. Also find out if the address is the primary address of the primary address type and set the DESC Oracle Object accordingly.

Function Level Description – BUILD_DETAIL_CHANGE_OBJECTS (local)

This private function builds a DESC Oracle Object to publish to the RIB for detail create and detail update messages (DTL_ADD, DTL_UPD). I_business_obj contains the header level key values (wh).

Function Level Description – BUILD_DETAIL_DELETE_OBJECTS (local)

This private function builds a REF Oracle Object to publish to the RIB for detail delete messages (DTL_DEL). I_business_obj contains the header level key values (wh).

Function Level Description – LOCK_THE_BLOCK (local)

This private function locks all queue records for the current business object (wh). This is to ensure that GETNXT and PUB_RETRY do 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 the queue table for DTL_UPD, DTL_DEL, and HDR_DEL messages.

Function Level Description – HANDLE_ERRORS (local)

This private procedure is called from GETNXT and PUB_RETRY when an exception is raised. I_seq_no is the sequence number of the driving WH_MFQUEUE record. I_function_keys contains detail level key values (wh, addr_key).

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
WHCre	WH Create Message	WHDesc.xsd
WHMod	WH Modify Message	WHDesc.xsd
WHDel	WH Delete Message	WHRef.xsd
WHDtlCre	WH Detail Create Message	WHDesc.xsd
WHDtlMod	WH Detail Modify Message	WHDesc.xsd
WHDtlDel	WH Detail Delete Message	WHRef.xsd
WHAddCre	WH Address Create	WHAddrDesc.xsd
WHAddMod	WH Address Modify	WHAddrDesc.xsd

Design 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.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
WH_MFQUEUE	Yes	Yes	Yes	Yes
WH_PUB_INFO	Yes	Yes	Yes	Yes
WH	Yes	No	No	No
ADDR	Yes	No	Yes	No
ADD_TYPE_MODULE	Yes	No	No	No

Work Orders In Publication API

Functional Area

Purchase Orders

Business Overview

A work order provides direction to a warehouse management system (such as RWMS) about work that needs to be completed on items contained in a recent purchase order. RMS publishes work orders soon after it publishes the purchase order itself. This is referred to as a 'work order in' message. This message is not to be confused with a 'work order out' message, which pertains to transfers.

Work order publication consists of a message containing attributes from the WO_DETAIL table plus the order number from the WO_HEAD table. One message is created each time a WO_DETAIL record is created, modified, or deleted. The primary key for the WO_DETAIL consists of the work order ID, warehouse, item, location, and sequence number. Thus, one work order can have multiple Work Order Create messages. When a WO_DETAIL record is created or modified, the message contains a full snapshot of the WO_DETAIL record. When a WO_DETAIL record is deleted, the message contains a partial snapshot of the WO_DETAIL record. Messages are retrieved from the message queue in the order they were created.

Work orders attached to purchase orders will have their messages published after the order has been published. Work orders attached to previously published, approved orders will have their messages published immediately.

Work orders are defined at the physical location level. The message family manager will send the warehouse at which the work order will be done. This is used by the RIB publication adaptor for routing messages to the appropriate warehouse.

Package Impact

Business Object ID

Work Order Id

Create

1. **Prerequisites**: An order has been distributed by item and location.
2. **Activity Detail**: A work order is ready to be published as soon as the order it is attached has been published. An initial publication message is made.
3. **Messages**: A "Work Order Create" message is queued. This message contains a snapshot of the attributes on the WO_DETAIL table.

Modify

1. **Prerequisites**: Work order has been created.
2. **Activity Detail**: The user is allowed to change attributes of the work order detail record. These changes are of interest to other systems and so this activity results in the publication of a message. Work orders attached to purchase orders will have their messages published after the order has been published. Work orders attached to previously published, approved orders will have their messages published immediately.
3. **Messages**: Any modifications to a work order detail record will cause a "Work Order Modify" message to be queued. This message contains the same attributes as the "Work Order Create" message.

Delete

1. **Prerequisites**: Work order has been created.
2. **Activity Detail**: Deleting a work order detail record removes it from the system. External systems are notified by a published message.
3. **Messages**: When a work order detail record is deleted a "Work Order Delete" message is queued. The message contains a partial snapshot of the WO_DETAIL table.

Package name: RMSMFM_WOIN

Spec file name: rmsmfm_woins.pls

Body file name: rmsmfm_woinb.pls

Package Specification – Global Variables

```
FAMILY      VARCHAR2(64)       'woin';
WO_ADD      CONSTANT           VARCHAR2(20)          'InBdWOCre';
WO_UPD      CONSTANT           VARCHAR2(20)          'InBdWOMod';
WO_DEL      CONSTANT           VARCHAR2(20)          'InBdWODel';
```

Function Level Description – ADDTOQ

```
Function: ADDTOQ
              (O_error_msg          OUT          VARCHAR2,
               I_queue_rec          IN           WOIN_MFQUEUE%ROWTYPE,
               I_publish_ind        IN           WO_DETAIL.PUBLISH_IND%TYPE)
```

This procedure is called by EC_TABLE_WDL_AIUDR, and takes a record type variable that consists of columns from the WO_DETAIL table and message type. It inserts a row into the message family queue WOIN_MFQUEUE along with the passed in values and the next sequence number from the message family sequence, and sets the status to unpublished. 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          OU           VARCHAR2,
               O_message_type       OU           VARCHAR2,
               O_message            OU           RIB_OBJECT,
               O_bus_obj_id         OU           RIB_BUSOBJID_TBL,
               O_routing_info       OU           RIB_ROUTINGINFO_TBL,
               I_num_threads        IN           NUMBER DEFAULT 1,
```

```
                I_thread_val          IN              NUMBER DEFAULT 1)
```

This publicly exposed procedure is typically called by a RIB publication adaptor. Its parameters are well defined and arranged in a specific order. The message type is the RIB defined short message name. Status code is one of five values. These codes are defined in the RIB_CODES package.

The error text parameter contains application-generated information, such as the application's sequence number of the message that failed, and the Oracle or other error that occurred when the retrieval failed.

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

Function Level Description – PROCESS_QUEUE_RECORD (local)

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 WOIN_QUEUE table is an insert or update (WO_ADD, WO_UPD):

- Builds the header object that contains work order ID and order number.
- Calls BUILD_DETAIL_OBJECTS to build the Oracle Object to publish to the RIB.

If the record from WOIN_QUEUE table is a delete (WO_DEL):

- Builds the header object that contains work order ID and order number.
- Calls BUILD_DETAIL_DELETE_OBJECTS to build the Oracle Object to publish to the RIB.

Function Level Description – BUILD_DETAIL_OBJECTS (local)

The function is responsible for building detail level Oracle Objects. It builds as many detail Oracle Object as it can given the passed in message type and business object key (work order ID).

Selects any details on the WOIN_MFQUEUE that are for the same work order ID and for the same message type.

- WOIN_MFQUEUE records that contain information being published are deleted.
- Each location represented in the published message is added to the ROUTING_INFO object.
- No more than the MAX_DETAILS_TO_PUBLISH numbers of records are put into Oracle Objects.

To avoid deleting information from the queue table that has not been published, deletes are accomplished using ROWIDs. All information is fetched using the same cursor; this ensures that the published message matches the deletes from the WOIN_MFQUEUE table regardless of trigger execution during GETNXT calls.

Function Level Description – BUILD_DETAIL_DELETE_OBJECTS (local)

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

Perform any BULK DML statements for deletion from WOIN_MFQUEUE.

Each location represented in the published message will be added to the ROUTING_INFO object.

Function Level Description – LOCK_THE_BLOCK (local)

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 the queue table for WO_DEL messages.

Function Level Description – ROUTING_INFO_ADD (local)

This function is called from within the BUILD_DETAIL_OBJECTS and BUILD_DETAIL_DELETE_OBJECTS. It will add the location from the message to the routing_info whenever a new location is added to the object being published.

Function Level Description – HANDLE_ERRORS (local)

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 WOIN_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 WO_DETAIL to capture Inserts, Updates, and Deletes.

Trigger name: EC_TABLE_WDL_AIUDR.TRG

Trigger file name: ec_table_wdl_aiudr.trg

Table: WO_DETAIL

This trigger will capture inserts/updates/deletes to the WO_DETAIL table and write data into the WOIN_MFQUEUE message queue.

- **Inserts:** Sends the header level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_WOIN.WO_ADD.
- **Updates:** Sends the header level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ WOIN.WO_UPD.
- **Deletes:** Sends the header level info to the ADDTOQ procedure in the MFM with the message type RMSMFM_ WOIN.WO_DEL.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
InBdWOCre	Work Order Create Message	WODesc.xsd
InBdWOMod	Work Order Modify Message	WODesc.xsd
InBdWODel	Work Order Delete Message	WORef.xsd

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
WOIN_MFQUEUE	Yes	Yes	No	Yes
WO_DETAIL	Yes	No	No	No
WOIN_MFQUEUE	Yes	Yes	Yes	Yes
WO_DETAIL	Yes	No	Yes	No

Design Assumptions

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

Work Orders Out Publication API

Functional Area

Transfers

Business Overview

This publication API facilitates the transmission of outbound work orders (OWO) from RMS to external systems. Only transfers that pass through a finisher before reaching the final location may be associated with work orders. The work orders are published upon approval of their corresponding transfers. The work order provides instructions for one or more of the following tasks to be completed at the finisher location:

- Perform some activity on an item, such as monogramming.
- Transform an item from one thing into another, such as dyeing a white t-shirt black.
- Combine bulk items into a pack or break down a pack into its component items.

Outbound work orders have their own message family because they cannot be bundled with transfer messages. This is because multi-legged transfers can be routed to either internal finishers (held as virtual warehouses) or external finishers (held as partners). Transfers to and from an internal finisher involve at least one book transfer. Because external systems may be unaware of virtual warehouses, book transfers are not communicated to external systems.

Outbound work order data is only published upon approval of the associated transfer. As such, *all* work order activity, transformation and packing data are contained in the same message. Because RMS does not allow users to modify work order activity, transformation or packing information for an approved transfer, detail-level messages of any type (create, delete, update) are never published. Outbound work order delete messages are published when the second leg of a multi-legged transfer is unapproved. This can be accomplished through the un-approval of an entire multi-legged transfer or the un-approval of the second leg only. A two-leg transfer that has had the first leg shipped can be set back to 'In Progress' status in order to make changes to the work order activities and the final location. When action has occurred, only the second leg is really set back to in progress. The first leg remains in shipped status.

Package Impact

Business Object ID

Transfer Work Order ID

Approve

1. **Prerequisites**: A multi-legged transfer must be approved and have work order details for each transfer detail.
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 and work order.
3. **Messages**: When a transfer with finishing is approved, an "outbdwocre" message is inserted into the queue. The family manager creates a hierarchical message containing a full snapshot of the transfer work order details at the time the message is published.

Delete

1. **Prerequisites**: The associated transfer has finishing and is being deleted.
2. **Activity Detail**: Deleting a transfer removes it, and the associated work order from the system. External systems are notified by a published Delete message that contains the number of the transfer work order to be deleted.
3. **Message**: When a transfer with finishing is deleted, an "outbdwodel", which is a flat notification message, is queued.

Unapproved

1. **Prerequisites:** A transfer with finishing is unapproved
2. **Activity Detail:** Not approving a transfer changes the status to input, which allows modification to the work order, transformation, packing, and item details. External systems are notified by a published Delete message that contains the number of the transfer work order to be deleted.
3. **Messages:** Not approving a transfer queues an "outbdwounaprv" request. This results in an "outbdwodel" message being published, which is a flat notification message.

Package name: RMSMFM_WOOUT

Spec file name: rmsmfm_woouts.pls

Body file name: rmsmfm_wooutb.pls

Package Specification – Global Variables

None

Function Level Description – ADDTOQ

```
Function: ADDTOQ
               (O_error_mesage        OUT     VARCHAR2,
                I_message_type        IN      VARCHAR2,
                I_tsf_wo_id           IN      tsf_wo_head.tsf_wo_id%TYPE)
```

There are some tasks relating to streamlining the queue clean up process that need to occur in ADDTOQ. The goal is to have at most one record on the queue for business transactions up until their initial publication.

- For header level insert messages (HDR_ADD), inserts a record in the WOOUT_PUB_INFO table. The work order number passed to the function should be inserted into the TSF_WO_ID column, and the published column should contain 'N'.
- If the business transaction has not been approved (woout_pub_info.publish_ind = 'N') and the triggering message is one of HDR_DEL and HDR_ANAPPRV, the record is not added to queue.

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)
```

This function fetches a record from the WOOUT_MFQUEUE table. The function fetches the record that has the lowest sequence number among queue records that have a pub_status of 'U' and a thread_no that matches the I_thread_val.

The LOCK_THE_BLOCK function is called. If it determines that WOOUT_MFQUEUE is locked for a particular work order, set the sequence limit local variable to the current sequence number. This will prevent the GETNXT function from attempting to lock and process the same work order message over and over again in the loop.

The WOOUT_MFQUEUE table is queried to determine if any records for the work order have been sent to the error hospital. If so, produce the 'SEND_TO_HOSP' error message and halt processing.

> **Note:** The only scenario in which a hospitalized record with the same tsf_wo_id as the message currently is processed would be found is if the initial HDR_ADD message had been hospitalized and a subsequent HDR_DEL or HDR_UNAPRV was being processed.

The PROCESS_QUEUE_RECORD function is called. If the break loop indicator returned from process_queue_record is TRUE, set the O_message_type output parameter to the message type fetched from the queue and return TRUE.

If the message type is null, the status code output parameter is set to API_CODES.NO_MSG. Otherwise, it is set to API_CODES.NEW_MSG and the O_bus_obj_id parameter is set to RIB_BUSOBJID_TBL(L_tsf_wo_id).

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);
```

This procedure is called from the RIB for woout_mfqueue.seq_no's that have been placed in the RIB's error hospital. It functions similarly to GETNEXT, except that it only fetches the record from WOOUT_MFQUEUE that contains the sequence number passed by the RIB.

If the message's tsf_wo_id is null, an API_CODES.NO_MSG error is raised. Then LOCK_THE_BLOCK is called. If the queue record is locked by another process, the status code is set to API_CODES.HOSPITAL. If the queue record is not locked by another process, PROCESS_QUEUE_RECORD is called. If the message returned from process_queue_record is null, the API_CODES.NO_MSG error is raised. Otherwise, if the message object is populated, it populates the business object table with the current work order number.

Function Level Description – PROCESS_QUEUE_RECORD (local)

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.

Check to see if the business object is being published for the first time. If the published_ind on the pub_info table is 'N', the business object is being published for the first time.

This function will set the O_break_loop parameter to FALSE in the following scenarios:

1. Processing a HDR_UNAPRV message for a work order that has a woout_pub_info.published of 'N'.
2. Processing a HDR_DEL message for a work order that has a woout_pub_info.published of 'N'.

The loop is not broken in these scenarios because they do not necessitate the publication of a message. Therefore, processing should continue so a message can be outputted.

If the message type is HDR_DEL and the work order has been published the function creates a work order ref object, and routing info object.

> **Note:** WO out routing info requires a 'to_loc' string and value.

If the message type is a HDR_UNAPRV and the work order has been published create a work order ref object and a routing info object. For all records associated with the work order on the tsf_wo_detail, tsf_xform_detail and tsf_packing tables, the publish_ind is set to 'N'.

> **Note:** A published value of 'I'n progress indicates that the work order was being published but it had more detail records than allowed for a single message. The maximum detail per message value can be found on the rib_settings table for each message family.

If the published indicator is 'N', the message type is set to HDR_ADD and the MAKE_CREATE function is called.

If the published indicator is 'I', the message type is set to DTL_ADD and the MAKE_CREATE function is called.

Function Level Description – MAKE_CREATE (local)

This function first calls the BUILD_HEADER_OBJECT function.

- It then calls the BUILD_DETAIL_OBJECTS function and updates the woout_pub_info column.
- It also updates the published_ind columns on TSF_WO_DETAIL, TSF_XFORM_DETAIL and TSF_PACKING.

Function Level Description – BUILD_HEADER_OBJECT (local)

This function fetches the transfer number and transfer parent number associated with the passed in work order number. It then calls the constructor for the rib_wooutdesc_rec, passing in the work order number, transfer number, and transfer parent number. Finally, it builds the routing info object.

Function Level Description – BUILD_DETAIL_OBJECTS (local)

The function is responsible for building detail level Oracle Objects. It builds as many detail Oracle Object as it can given the passed in message type and business object keys.

If the function is being called from MAKE_CREATE:

- Selects any unpublished detail records from the business transaction (tsf_wo_detail, tsf_xfrom_detail, tsf_packing).
  - Ensures that WOOUT_MFQUEUE is deleted from as needed. If there is more than one WOOUT_MFQUEUE record for a detail level record, it makes sure they all get deleted. Current state should be considered, not every change.
  - Ensures that ROUTING_INFO is constructed if routing information is stored at the detail level in the business transaction.
  - Ensures that no more than MAX_DETAILS_TO_PUBLISH records are put into Oracle Objects.
  - Ensures that the detail records being added to the object have not already been published. This can happen if GETNXT was previously called for the current business object, and the MAX_DETAILS_TO_PUBLISH limit had been reached.

Function Level Description – DELETE_QUEUE_REC (local)

This function deletes a record from the outbound work order queue table based on a passed-in sequence number.

Function Level Description – BUILD_WODTL_OBJECT (local)

This function fetches the activity_id, unit_cost and comments for all records from tsf_wo_detail containing the passed in item and work order ID. For each record found:

Populates the wooutactivity record with the activity_id, unit_cost and comments. Then, adds the wooutactivity record to the wooutactivity table.

After all details are processed, the WOOUTACTIVITY table is added to the wooutdtl record that was passed into the function.

Function Level Description – BUILD_PACKING_OBJECT (local)

```
Procedure: BUILD_PACKING_OBJECT
    (O_error_msg              IN OUT         VARCHAR2,
     O_packing_message        IN OUT         nocopy RIB_WOOUTPACKING_TBL,
     IO_rib_wooutpacking_rec  IN OUT         nocopy RIB_WOOUTPACKING_REC,
     I_tsf_packing_id         IN             tsf_packing.tsf_packing_id%TYPE))
```

This function first constructs the "RIB_WOOutpackFrom_REC" object by fetching tsf_packing_detail.item where the tsf_packing_id matches that which was passed into the function and the record_type is 'F' (from). Once complete, adds the WOOUTPACKFROM table to the wooutpacking_rec passed to the function.

Next, the "RIB_WOOutpackTo_REC" object is constructed. Fetches the tsf_packing_detail.item where the tsf_packing_id matches that which was passed into the function and the record_type is 'R' (result). Once complete, adds the WOOUTPACKTO table to the wooutpacking_rec passed to the function.

Function Level Description – LOCK_THE_BLOCK (local)

The function locks all records on the queue table for the business object. It has an O_queue_locked output that specifies whether some process other than the current process has the queue locked.

Function Level Description – HANDLE_ERRORS (local)

This procedure handles error status values of 'H'ospital. If the LP_error_status value is 'H'ospital, it populates the business object table with the current work order number, then creates a routing info object and populates it with the sequence number of the queue record. Finally a WOOutRef object is created and added to the O_message object.

The woout_mfqueue is updated by setting the pub_status equal to API_CODES.HOSPITAL.

Trigger Impact

A trigger on the WO_DETAIL and TSF_HEAD exists to capture Inserts, Updates, and Deletes.

Trigger file name: ec_table_thd_aiudr.trg

Table: TSFHEAD

- **Inserts:** Sends the tsf_wo_id level info to the RMSMFM_WOOUT.ADDTOQ procedure in the MFM with the message type RMSMFM_WOOUT.HDR_ADD.

Updates:

- Sends the tsf_wo_id level info to the RMSMFM_WOOUT.ADDTOQ procedure in the MFM with the message type RMSMFM_WOOUT.HDR_UNAPRV.
- When a transfer is placed in 'A'pproved status the message type for this action will be outbdwocre. When a transfer's status is updated to 'D'eleted, the family manager inserts a record into the queue with a message_type = outbdwodel. When the status is set to 'I'nput from Approved, the family manager inserts a record into the queue with message type = outbdwounaprv.
- **Deletes:** Sends the level info to the RMSMFM_WOOUT.ADDTOQ procedure in the MFM with the message type RMSMFM_WOOUT.HDR_DEL.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
OutBdWoCre	Work Order Create Message	WODesc.xsd
OutBdWoDel	Work Order Delete Message	WORef.xsd

Design Assumptions

- The order upon which transfer and work order messages arrive at locations participating in a multi-legged transfer does not need to be programmatically controlled.
- Work order information is never published solely at a detail level. That is, insertions, deletions and updates to work order records may not happen once the work order has been approved. In order to modify work order information, the user will need to unapprove the associated transfer. This will cause a work order header delete message to be published.

- When a work order is unapproved or deleted, header level reference information only can be published. Reference information at the detail level is not required to be published, because work order publication is never done at the individual detail level.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
WOOUT_MFQUEUE	Yes	Yes	Yes	Yes
WOPUT_PUB_INFO	Yes	Yes	Yes	Yes
TSFHEAD	Yes	No	No	No
TSF_WO_HEAD	Yes	No	No	No
TSF_WO_DETAIL	Yes	No	Yes	No
TSF_XFORM	Yes	No	No	No
TSF_XFORM_DETAIL	Yes	No	Yes	No
TSF_PACKING	Yes	No	Yes	No
TSFDETAIL	Yes	No	No	No
TSF_PACKING_DETAIL	Yes	No	No	No

3

Subscription Designs

This chapter provides an overview of the Subscription APIs used in the RMS environment and various functional attributes used in the APIs.

For more information on RIB_XML, see Appendix: RIB_XML.

Allocation Subscription API

Functional Area

Allocation

Business Overview

The allocation subscription API allows an external application to create, update, and delete allocations within RMS. The main reason for doing so is to successfully interface and track all dependent bills of lading (BOL) and receipt messages into RMS, as well as to calculate stock on hand correctly.

The allocation subscription API can be used by a 3rd party merchandise system to create, update and delete allocations based on warehouse inventory or cross-dock. The Oracle Retail Allocation product does NOT use this API to interface allocations to RMS. From an Oracle Retail perspective, this API is used by AIP to support the creation of cross dock POs, based on POs sent to RMS using the Order Subscription API.

Allocations only involve stockholding locations. This includes the ability to process allocations to both company and franchise stores, as well as any stockholding warehouse location, excepting internal finishers. If an allocation for a franchise store is received, RMS will also create a corresponding franchise order. This API supports either warehouse-to-warehouse or warehouse-to-store allocations, but no mix-match in a single allocation.

Allocation details can be created, edited, or deleted within the allocation message. Detail line items must exist on an allocation header create message for an allocation to be created. New item location relationships will be created for allocation detail line items entering RMS that do not previously exist within RMS.

New locations can be added to existing allocations, or current locations can be modified on existing allocations. If modifying an existing location, RMS assumes the passed in quantity is an adjustment to the current quantity as opposed to an over write. For example, if the current qty_allocated on ALLOC_DETAIL is 10, and a detail modification message for the same item contains a qty_allocated of 8, ALLOC_DETAIL will be updated with qty_allocated of 10+8 =18.

Details can be individually removed from an allocation if the detail is not in-transit or received or in progress. An entire allocation can be deleted if none of details are in-transit or received or in progress.

Package Impact

Filename: rmssub_xallocs/b.pls

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

```
I_message_type        IN              VARCHAR2)
```

This procedure needs to initially ensure that the passed in message type is a valid type for Allocation messages. If the message type is invalid, a status of "E" will be returned to the external system along with an appropriate error message informing the external system that the status is invalid.

If the message type is valid, the generic RIB_OBJECT needs to be downcast to the actual object using Oracle's treat function. If the downcast fails, a status of "E" will 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 calls the RMSSUB_XALLOC_VALIDATE.CHECK_MESSAGE function to determine whether the message is valid. If the message passed RMS business validation, the function returns true, otherwise it returns false. If the message has failed RMS business validation, a status of "E" will 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 will be persisted to the RMS database. It calls the RMSSUB_XALLOC_SQL.PERSIST_MESSAGE() function. If the database persistence fails, the function returns false. A status of "E" will be returned to the external system along with the error message returned from the PERSIST_MESSAGE() function.

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

RMSSUB_XALLOC.HANDLE_ERROR() is the standard error handling function that wraps the API_LIBRARY.HANDLE_ERROR function.

Filename: rmssub_xallocvals/b.pls

```
RMSSUB_XALLOC_VALIDATE.CHECK_MESSAGE
               (O_error_message       IN OUT         VARCHAR2,
                O_alloc_rec           OUT            ALLOC_REC,
                I_message             IN             RIB_XAllocDesc,
                I_message_type        IN             VARCHAR2)
```

This function performs all business validation associated with message and builds the allocation record for persistence.

Note: Some of the business validation is referential or involves uniqueness. This validation is handled automatically by the referential integrity constraints and the unique index implemented on the database and is not described below.

Allocation Create

- Check required fields.
- If item is a pack, verify receive as type is Pack for from location (warehouse).
- Verify details exist.
- Default fields (status at header, qty pre-scaled, non scale ind).
- Build allocation records.
- Perform following steps if allocation is **not** cross-docked from an order.
  - Retrieve and build all to-locations that the item does not currently exist at.
  - Build price history records.

Allocation Modify

- Check required fields
- Populate record.

Allocation Delete

- Check required fields
- Verify the allocation is not in-transit or received or in progress. An allocation in progress will have processed_ind equal to 'Y'. An allocation in-transit or received will have a value (other than zero) for any of the following fields: distro quantity, selected quantity, canceled quantity, received quantity, or PO received quantity.

Allocation Detail Create

- Check required fields
- Verify details exist.
- Build allocation records.
- Perform following steps if allocation is NOT cross-docked from an order.
  - Retrieve and build all to-locations that the item does not currently exist at.
  - Build price history records.

Allocation Detail Modify

- Check required fields.
- If existing allocation records are being modified:
  - Verify the allocation is not in-transit or received or in progress.
  - Verify modification to quantity does not fall to zero or below.

Allocation Detail Delete

- Check required fields.
- Verify the allocation is not in-transit or received or in progress.
- Check if deleting detail(s) removes all records from allocation. If so, process message as allocation delete.

Filename: rmssub_xallocsqls/b.pls

```
RMSSUB_XALLOC_SQL.PERSIST
               (O_error_message       IN OUT         VARCHAR2,
                I_dml_rec             IN             ALLOC_RECTYPE ,
                I_message             IN             RIB_XAllocDesc)
```

Allocation Create

- Insert a record into the allocation header table.
- Insert a record into the allocation detail table.
- Insert a record into the allocation charge table.
- Insert records into the franchise order tables, if allocating to franchise stores.
- For an approved non-cross dock allocation, update transfer reserved for from-location. If a pack item is allocated from a warehouse with pack receive_as_type of 'P' – pack, also update pack component reserved qty for the from-location.
- For an approved non-cross dock allocation, update transfer expected for to-location. If a pack item is allocated to a warehouse with pack receive_as_type of 'P' – pack, also update pack component expected qty for the to-location.

- If item is not ranged to the to-location, call NEW_ITEM_LOC to create item-location on the fly with ranged_ind of 'Y'. This will insert a record into ITEM_LOC, ITEM_LOC_SOH, ITEM_SUPP_COUNTRY_LOC, PRICE_HIST tables and put a new item-loc event on the future cost event queue. For Brazil localized, item country relationship must exist for the item-location being created.

Allocation Modify

- Update header record (alloc desc and release date).

Allocation Detail Create

- Same as Allocation Create, except that there is no need to insert into ALLOC_HEADER table.

Allocation Detail Modify

- Update the allocation detail table by adjusting the existing allocated quantity using the passed in quantity. This can either increase or decrease the existing quantity.
- Update franchise order quantity if allocating to franchise stores.
- For an approved non-cross dock allocation, update transfer reserved for from-location. If a pack item is allocated from a warehouse with pack receive_as_type of 'P' – pack, also update pack component reserved qty for the from-location.
- For an approved non-cross dock allocation, update transfer expected for to-location. If a pack item is allocated to a warehouse with pack receive_as_type of 'P' – pack, also update pack component expected qty for the to-location.

Allocation Detail Delete

- Delete the record from the allocation detail table.
- Delete the record from the allocation charge table.
- Delete records from the franchise order tables if the details deleted involve franchise stores.
- If deleting details from an approved non-cross dock allocation, update transfer reserved for from-location. If a pack item is allocated from a warehouse with pack receive_as_type of 'P' – pack, also update pack component reserved qty for the from-location.
- If deleting details from an approved non-cross dock allocation, update transfer expected for to-location. If a pack item is allocated to a warehouse with pack receive_as_type of 'P' – pack, also update pack component expected qty for the to-location.

Allocation Delete

- Update the allocation header to Cancelled ('C') status.
- Update the linked franchise order to Cancelled ('C') status.
- Delete all associated record from the allocation charge table.
- If deleting an approved non-cross dock allocation, update transfer reserved for from-location. If a pack item is allocated from a warehouse with pack receive_as_type of 'P' – pack, also update pack component reserved qty for the from-location.
- If deleting an approved non-cross dock allocation, update transfer expected for to-location. If a pack item is allocated to a warehouse with pack receive_as_type of 'P' – pack, also update pack component expected qty for the to-location.

Message XSD

Here are the filenames that correspond with each message type. Refer the mapping documents for each message type ofr details about composition of each message.

Message Type	Message Type Description	XML Schema Definition (XSD)
XAllocCre	External Allocation Create	XAllocDesc.xsd
XAllocDel	External Allocation Delete	XAllocRef.xsd
XAllocDtlCre	External Allocation Detail Create	XAllocDesc.xsd
XAllocDtlDel	External Allocation Detail Delete	XAllocRef.xsd
XAllocDtlMod	External Allocation Detail Modification	XAllocDesc.xsd
XAllocMod	External Allocation Modification	XAllocDesc.xsd

Design Assumptions

- This API only applies to store level zone pricing.
- This API does not currently handle inner packs when needing to create pack component location information.
- Passed in item is at transaction level.
- From location is a non-finisher stockholding warehouse (i.e. a virtual warehouse).
- Because the allocation quantities are not generated based upon RMS inventory positions, RMS provides no stock on hand or inventory validation.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ALLOC_HEADER	Yes	Yes	Yes	No
ALLOC_DETAIL	Yes	Yes	Yes	Yes
ALLOC_CHRG	Yes	Yes	No	Yes
ITEM_SUPP_COUNTRY	Yes	No	No	No
ITEM_SUPP_COUNTRY_LOC	Yes	Yes	No	No
ITEM_LOC_SOH	Yes	Yes	Yes	No

TABLE	SELECT	INSERT	UPDATE	DELETE
STORE	Yes	No	No	No
WH	Yes	No	No	No
ITEM_LOC	Yes	Yes	No	No
SYSTEM_OPTIONS	Yes	No	No	No
ORDHEAD	Yes	No	No	No
PRICE_HIST	No	Yes	No	No
ITEM_MASTER	Yes	No	No	No
WF_ORDER_HEAD	Yes	Yes	Yes	No
WF_ORDER_DETAIL	Yes	Yes	Yes	No
WF_ORDER_EXP	Yes	Yes	Yes	No
WF_CUSTOMER	Yes	No	No	No
WF_CUSTOMER_GROUP	Yes	No	No	No
WF_COST_RELATIONSHIP	Yes	No	No	No
WF_COST_BUILDUP_TMPL_HEAD	Yes	No	No	No
WF_COST_BUILDUP_TMPL_DETAIL	Yes	No	No	No
FUTURE_COST	Yes	No	No	No

Appointments Subscription API

Functional Area

Appointments

Business Overview

An appointment is information about the arrival of merchandise at a location. From the RIB, RMS subscribes to appointment messages that are published by an external application, such as a warehouse management system (for example, RWMS). RMS processes these messages and attempts to receive against and close out the appointment. In addition, RMS attempts to close the document that is related to the appointment. A document can be a purchase order, a transfer, or an allocation.

Appointment status

Appointment messages cause the creation, update, and closure of an appointment in RMS. Typically the processing of a message results in updating the status of an appointment in the APPT_HEAD table's status column. Valid values for the status column include:

- SC–Scheduled
- MS–Modified Scheduled
- AR–Arrived
- AC–Closed

A description of appointment processing follows.

Appointment processing

The general appointment message processes occur in this order:

1. An appointment is created for a location with a store or warehouse type from a scheduled appointment message. It indicates that merchandise is about to arrive at the location. Such a message results in a 'SC' status. At the same time, the APPT_DETAIL table is populated to reflect the purchase order, transfer, or allocation that the appointment corresponds to, along with the quantity of the item scheduled to be sent.
2. Messages that modify the earlier created appointment update the status to 'MS'.
3. Once the merchandise has arrived at the location, the appointment is updated to an 'AR' (arrived) status.
4. Another modification message that contains a receipt identifier prompts RMS to insert received quantities into the APPT_DETAIL table.
5. After all items are received, RMS attempts to close the appointment by updating it to an 'AC' status.
6. RMS will close the corresponding purchase order, transfer, or allocation 'document' if all appointments are closed.

Appointment records indicate the quantities of particular items sent to various locations within the system. The basic functional entity is the appointment record. It consists of a header and one or more detail records. The header is at the location level; the detail record is at the item-location level (with ASN as well, if applicable). Documents are stored at the detail level; a unique appointment ID is stored at the header level. In addition, a receipt number is stored at the detail level and is inserted during the receiving process within RMS.

Package Impact

Filename: rmssub_receivings/b.pls

```
RMSSUB_RECEIVING.CONSUME
               (O_status_code        IN OUT        VARCHAR2,
                O_error_message      IN OUT        VARCHAR2,
                I_message            IN            RIB_OBJECT,
                I_message_type       IN            VARCHAR2,
                O_rib_otbdesc_rec    OUT           "RIB_OTBDesc_REC",
                O_rib_error_tbl      OUT           RIB_ERROR_TBL)
```

This is the procedure called by the RIB. This procedure will make calls to receiving or appointment functions based on the value of I_message_type. If I_message type is RECEIPT_ADD, RECEIPT_UPD, or RECEIPT_ORDADD, then a call is made to RMSSUB_RECEIPT.CONSUME, casting the message as a RIB_RECEIPTDESC_REC. If I_message_type is APPOINT_HDR_ADD, APPOINT_HDR_UPD, APPOINT_HDR_DEL, APPOINT_DTL_ADD, APPOINT_DTL_UPD, or APPOINT_DTL_DEL, then a call is made to RMSSUB_APPOINT.CONSUME.

> **Note**: The receiving process RMSSUB_RECEIPT.CONSUME is described in a separate Receiving Subscription API document.

```
RMSSUB_RECEIVING.HANDLE_ERRORS
               (O_status_code         IN OUT         VARCHAR2,
                IO_error_message      IN OUT         VARCHAR2,
                I_cause               IN             VARCHAR2,
                I_program             IN             VARCHAR2)
```

Standard error handling function that wraps the API_LIBRARY.HANDLE_ERROR function.

Filename: rmssub_appoints/b.pls

```
RMSSUB_APPOINT.CONSUME.CONSUME
               (O_status_code         IN OUT         VARCHAR2,
                O_error_message       IN OUT         VARCHAR2,
                I_message             IN             RIB_OBJECT,
                I_message_type        IN             VARCHAR2)
```

This function validates that the message type is valid for appointment subscription. If not, it returns a status of 'E' - Error along with an error message to the calling function.

If it is valid, it casts the message as "RIB_APPOINTDESC_REC" for create and modification message types (APPOINT_HDR_ADD, APPOINT_HDR_UPD, APPOINT_DTL_ADD, APPOINT_DTL_UPD), or "RIB_APPOINTREF_REC" for delete message types (APPOINT_HDR_DEL, APPOINT_DTL_DEL). It then calls local procedures HDR_ADD_CONSUME, HDR_UPD_CONSUME, HDR_DEL_CONSUME, DTL_ADD_CONSUME, DTL_UPD_CONSUME and DTL_DEL_CONSUME to perform the actual subscription logic.

Appointment Create

- Location must be a valid store or warehouse.
- Document must be valid based on document type ('P' for purchase order, 'T', 'D', 'V' for transfer, 'A' for allocations).
- Item must be a valid item.
- Insert header to APPT_HEAD if a record does not exist; otherwise, the header insert is skipped.
- Insert details to APPT_DETAIL if records do not already exist. Details that already exist are skipped.

Appointment Modify

- Location must be a valid store or warehouse.
- Item must be a valid item.
- Update or insert into APPT_HEAD. Call APPT_DOC_CLOSE_SQL.CLOSE_DOC to close the document if the new appointment status is 'AC'.

Appointment Delete

- Location must be a valid store or warehouse.
- Delete both header and detail records in APPT_HEAD and APPT_DETAIL.

Appointment Detail Create

- Location must be a valid store or warehouse.
- Document must be valid based on document type ('P' for purchase order, 'T', 'D', 'V' for transfer, 'A' for allocations).
- Item must be a valid item.
- Insert details to APPT_DETAIL if records do not already exist. Details that already exist are skipped.

Appointment Detail Modify

- Location must be a valid store or warehouse.
- Update or insert into APPT_DETAIL.

Appointment Detail Delete

- Location must be a valid store or warehouse.
- Delete from APPT_DETAIL.

Message XSD

Here are the filenames that correspond with each message type. Please see *RIB documentation* for each message type in order to get a detailed picture of the composition of each message.

Message Types	Message Type Description	XML Schema Definition (XSD)
Appointcre	Appointment Create Message	AppointDesc.xsd
Appointhdrmod	Appointment Header Modify Message	AppointDesc.xsd
Appointdel	Appointment Delete Message	AppointRef.xsd
Appointdtlcre	Appointment Detail Create Message	AppointDesc.xsd
Appointdtlmod	Appointment Detail Modify Message	AppointDesc.xsd
Appointdtldel	Appointment Detail Delete Message	AppointRef.xsd

Design Assumptions

- The adaptor is only set up to call stored procedures, not stored functions. Any public program needs to be a procedure.
- Detail records may contain the same PO/item combination, differentiated only by the ASN number; however, the ASN field will be NULL for detail records which are not associated with an ASN.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
APPT_HEAD	Yes	Yes	Yes	Yes
APPT_DETAIL	Yes	Yes	Yes	Yes
ORDHEAD	Yes	No	Yes	No
TSFHEAD	Yes	No	Yes	No

TABLE	SELECT	INSERT	UPDATE	DELETE
ALLOC_HEADER	Yes	No	Yes	No
STORE	Yes	No	No	No
WH	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
ORDLOC	Yes	No	No	No
DEAL_CALC_QUEUE	Yes	No	No	Yes
OBLIGATION	Yes	No	No	No
OBLIGATION_COMP	Yes	No	No	No
ALC_HEAD	Yes	No	No	Yes
ALC_COMP_LOC	Yes	No	No	Yes
V_PACKSKU_QTY	Yes	No	No	No
TSFDETAIL	Yes	No	No	No
SHIPMENT	Yes	No	No	No
SHIPSKU	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
ITEM_LOC_SOH	Yes	No	Yes	No
ALLOC_DETAIL	Yes	No	No	No

ASNIN Subscription API

Functional Area

Advance shipping notice (ASN) from a supplier

Business Overview

A supplier or consolidator will send an advanced shipping notice (ASN) to RMS through the Oracle Retail Information Bus (RIB). RMS subscribes to the ASN information and places the information onto RMS tables depending upon the validity of the records enclosed within the ASN message.

The ASN message will consist of a header record, a series of order records, carton records, and item records. For each message, header, order and item record(s) will be required. The carton portion of the record is optional. If a carton record is present, however, then that carton record must contain items in it.

The header record will contain information about the shipment as a whole. The order records will identify which orders are associated with the merchandise being shipped. If the shipment is packed in cartons, carton records will identify which items are in which cartons. The item records will contain the items on the shipments, along with the quantity shipped. The items on the shipment should be on the ORDLOC table for the order and location specified in the header and order records.

The location that is contained on the ASN will represent the expected receiving location for the order. If the location is a non-stockholding store in RMS, then the shipment will also be automatically received when the ASN is processed. Two types of non-stockholding stores orders are supported in this integration – franchise stores and drop ship customer orders.

Package Impact

Filename: rmssub_asnins/b.pls

```
RMSSUB_ASNIN.CONSUME
                (O_STATUS_CODE        IN OUT        VARCHAR2,
                O_ERROR_MESSAGE       IN OUT        VARCHAR2,
                I_MESSAGE             IN            RIB_OBJECT,
                I_MESSAGE_TYPE        IN            VARCHAR2);
```

The following is a description of the RMSSUB_ASNIN.COMSUME procedure:

1. The public procedure checks if the message type is create (ASNINCRE), modify (ASNINMOD), or delete (ASNINDEL).
2. If the message type is ASNINDEL then,
   - It will cast the message to type "RIB_ASNInRef_REC".
   - If a message exists in the record then it will call the private function PROCESS_DELETE to delete the ASN record from the appropriate shipment and invoice database tables depending upon the success of the validation.
   - If no messages exist in the record then it will raise a program error that no message was deleted.
3. If the message type is ASNINCRE or ASNINMOD then:
   - It will cast the message to type "RIB_ASNInDesc_REC".
   - It will parse the message by calling the private function PARSE_ASN.
   - After parsing the message, it will check if the message contains a PO record. A program error will be raised if either the message type is invalid, or if there is no PO record.
   - If the records are valid after parsing, the detail records are retrieved and processed in a loop.

   Inside the loop:

   a. Records are passed on to the private function PARSE_ORDER.
   b. Delete container and item records from the previous order.
   c. Check if CARTON_IND is equal to 'C'.
   d. If CARTON_IND equal to 'C', call private functions PARSE_CARTON and PARSE_ITEM to parse cartons and items within a carton.
   e. If CARTON_IND is NOT equal to 'C', call private function PARSE_ITEM to parse items that are not part of a container.
   f. Call private function PROCESS_ASN with parsed data on ASN, order, carton, and item records. The records are place in the appropriate shipment and ordering database tables depending upon the success of the validation.

Error Handling

If an error occurs in this procedure or any of the internal functions, this procedure places a call to **HANDLE_ERRORS** in order to parse a complete error message and pass back a status to the RIB.

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

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

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

Private Internal Functions and Procedures

PARSE_ASN

This function will be used to extract the header level information from "RIB_ASNInDesc_REC" and place that information onto an internal ASN header record.

```
TYPE asn_record IS RECORD(asn                    SHIPMENT.ASN%TYPE,
                          destination            SHIPMENT.TO_LOC%TYPE,
                          ship_date              SHIPMENT.SHIP_DATE%TYPE,
                          est_arr_date           SHIPMENT.EST_ARR_DATE%TYPE,
                          carrier                SHIPMENT.COURIER%TYPE,
                          ship_pay_method        ORDHEAD.SHIP_PAY_METHOD%TYPE,
                          inbound_bol            SHIPMENT.EXT_REF_NO_IN%TYPE,
                          supplier               ORDHEAD.SUPPLIER%TYPE,
                          carton_ind             VARCHAR2(1));
```

PARSE_ORDER

This function will be used to extract the order level information from "RIB_ASNInPO_REC" and ASN number from shipment table, and place that information onto an internal order record.

PARSE_CARTON

This function will be used to extract the carton level information from "RIB_ASNInCtn_REC" and ASN and ORDER number from shipment table, and place that information onto an internal carton record.

PARSE_ITEM

This function will be used to extract the item level information from "RIB_ASNInItem_REC", ASN and ORDER number in the shipment table, and CARTON number from carton table, and place that information onto an internal item record.

Validation

PROCESS_ASN

After the values are parsed for a particular order in an ASN record, RMSSUB_ASNIN.CONSUME will call this function, which will in turn call various functions inside ASN_SQL in order to validate the values and process the ASN depending upon the success of the validation.

Only one ASN and order record will be passed in at a time, whereas multiple cartons and items will be passed in as arrays into this function. If one order, carton or item value is rejected, then current functionality dictates that the entire ASN message will be rejected.

PROCESS_DELETE

In the event of a delete message, this function will be called rather than PROCESS_ASN. This function will take the asn_no from the parsing function and pass it into ASN_SQL in order to delete the ASN record from the appropriate shipment and invoice tables. A received shipment cannot be deleted.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
asnincre	ASN Inbound Create Message	ASNInDesc.xsd
asnindel	ASN Inbound Delete Message	ASNInRef.xsd
asninmod	ASN Inbound Modify Message	ASNInDesc.xsd

Design Assumptions

None

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
SHIPMENT	Yes	Yes	Yes	Yes
SHIPSKU	Yes	Yes	No	Yes
CARTON	No	Yes	No	Yes
INVC_XREF	No	No	No	Yes
STORE	Yes	No	No	No
WH	Yes	No	No	No
ORDHEAD	Yes	No	No	No

ASNOUT Subscription API

Functional Area

ASNOUT

Business Overview

An internal advance shipment notification (ASN) message holds data that is used by RMS to create or modify a shipment record. Also known as a bill of lading (BOL), internal ASNs are published by an application that is external to RMS, such as a store system (SIM, for example) or a warehouse management system (RWMS, for example). In contrast to a BOL is the external ASN, which is generated by a supplier and shows merchandise movement from the supplier to a retailer location, like a warehouse or store. This overview describes the BOL type of advance shipment notification. For external ASN from suppliers, see ASNIN Subscription API.

Internal ASNs are a notification to RMS that inventory is moving from one location to another. RMS subscribes to BOL messages from the Oracle Retail Integration Bus (RIB).

The external application publishes these ASN messages for:

- Pre-existing allocations.
- Pre-existing transfers.
- Externally generated transfers, created in the store or warehouse (created as transfer type of 'EG' within RMS).

Individual stock orders are held on the transfer and allocation header tables in RMS. A message may contain data about multiple transfers or allocations, and as a result, the shipment record in RMS would reflect these multiple movements of merchandise. A bill of lading number on the shipment record is a means of tracking one or more transfers and allocations back through the respective stock order records.

This API also supports shipment notification for customer order transfers. There are two special handlings of these shipment notifications:

- When store inventory is used to fulfill a customer request, SIM will send an ASNOut message without a ship-to location. In that case, RMS will ignore these ASNOut messages, as these are not associated with a transfer or allocation in RMS.
- When a warehouse directly ships to a customer, RWMS will send an ASNOut message with a virtual store as the ship-to location. In that case, RMS will auto-receive the shipment.

Additionally, this API supports shipment notifications for franchise order and return transactions. Shipping of franchise orders to a stockholding franchise store, as well as shipping of franchise returns from a stockholding franchise store, is managed in a similar way as a regular store transaction, except that different transaction codes are used for TRAN_DATA. Shipping of franchise orders to a non-stockholding franchise store from a warehouse or a company store will be auto-received in RMS when the ASN is processed.

L10N Localization Decoupling Layer

This is a layer of code which enables decoupling of localization logic that is only required for certain country-specific configuration. This layer affects the RIB API flows including ASNOut subscription. This allows RMS to be installed without requiring customers to install or use this localization functionality, where not required.

BOL Message Structure

Because RMS uses a BOL message only to create a new shipment record, there is one subscribed BOL message. The message consists of a header, a series of transfers or

allocations (called 'Distro' records), carton records, and item records. Thus the structure of a BOL hierarchical message would be:

- Message header–This is data about the entire shipment including all distro records, cartons, and items.
- Distro record–The individual transfer or allocation being shipped.
- Carton–Carton numbers and location, as well as carton records will identify which items are in which cartons.
- Items– Details about all items in the carton, including shipped quantity.

When external locations (stores or warehouses) ship products, they send a BOL message (otherwise known as an outbound ASN message) to let RMS know that they are shipping the stock and to let the receiving locations know that the stock is on the way. The external locations can create BOL messages for three scenarios: a transfer was requested (RMS knows about it), an allocation was requested (RMS knows about it), and on their own volition (externally generated - EG). A single BOL message can contain records generated for any or all of these transactions.

RMS allows multiple transfers and allocations per shipment, which supports the operational process whereby a stock order shipment is often a group of transfers or allocations on one truck. These transfers or allocations are grouped together using a single BOL number. This number will be stored on the header record for the shipment. All shipments will be associated with a BOL number.

Package Impact

Filename: rmssub_asnouts/b.pls

```
RMSSUB_ASNOUT.CONSUME
              (O_status_code        IN OUT        VARCHAR2,
               O_error_message      IN OUT        VARCHAR2,
               I_message            IN            RIB_OBJECT,
               I_message_type       IN            VARCHAR2)
```

This procedure will initially ensure that the passed in message type is a valid type for ASNOUT messages.

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 is downcast to the actual object using the Oracle's treat function. If the downcast fails, a status of "E" is returned to the external system along with an appropriate error message informing the external system that the object passed in is invalid.

If the downcast is successful, then consume will parse the message, verify that the message passes all of RMS's business validation and persist the information to the RMS database. It does this by calling CONSUME_SHIPMENT.

```
RMSSUB_ASNOUT.CONSUME_SHIPMENT
```

- (O_status_code IN OUT VARCHAR2,
- O_error_message IN OUT VARCHAR2,
- I_message IN RIB_OBJECT,
- I_message_type IN VARCHAR2,
- I_check_l10n_ind IN VARCHAR2)

- Perform localization check. If localized, invoke RFM's logic through L10N_SQL decoupling layer for procedure key 'CONSUME_SHIPMENT'. If not localized, call CONSUME_SHIPMENT for normal processing.
-

```
RMSSUB_ASNOUT.CONSUME_SHIPMENT
               (O_error_message     IN OUT  VARCHAR2,
                IO_L10N_RIB_REC     IN OUT  L10N_OBJ)
```

- Public function to call RMSSUB_ASNOUT.CONSUME_SHIPMENT_CORE.

```
RMSSUB_ASNOUT.CONSUME_SHIPMENT_CORE
               (O_error_message     IN OUT  VARCHAR2,
                I_message           IN      RIB_OBJECT,
                I_message_type      IN      VARCHAR2)
```

This function contains the main processing logic:

- Calls PARSE_BOL to parse the shipment level information on the message. Insert or update shipment based on the bill of lading number (bol_nbr).
- One shipment can contain multiple distros (transfers and allocations in RMS). Within each distro, call PARSE_DISTRO and PARSE_ITEM to parse and build a collection of items that are transferred or allocated.
- For break-to-sell items, if the sellable item is on the message, call CHECK_ITEMS and GET_ORDERABLE_ITEMS to convert the sellable item(s) to the corresponding orderable item(s). The orderable items will be inserted or updated on transfer/allocation and shipment tables.
- For catch weight items, validate and aggregate weight for the same item.
- Call PROCESS_DISTRO to perform business logic associated with shipping a transfer or an allocation, including insert or update transfer/allocation header and detail, insert or update SHIPSKU, move inventory to in transit buckets on ITEM_LOC_SOH, write stock ledger.
- Bulk inserts and updates are performed to improve performance.

If an error occurs in the process, a status of 'E' is returned to the external system along with the failure message. Otherwise, a success status, 'S', is returned to the external system indicating that the message has been successfully received and persisted to the RMS database.

PARSE_BOL

This function parses the "RIB_ASNOutDesc_Rec" and builds an API bol_record for processing. It also calls RMSSUB_ASNOUT.PROCESS_BOL to check the existence of SHIPMENT based on the bol number.

PROCESS_BOL

This function calls BOL_SQL.PUT_BOL to check the existence of SHIPMENT based on the BOL number.

PARSE_DISTRO

This function parses the "RIB_ASNOutDesc_Rec" and builds an API distro_record for processing.

PARSE_ITEM

This function builds a collection of API item_table that contains item level information for the transfer or allocation. For a simple pack catch weight item, it also aggregates the weight for the same item.

PROCESS_DISTRO

Depending on the distro type (transfer or allocation), this function calls BOL_SQL.PUT_TSF, BOL_SQL.PUT_TSF_ITEM, and BOL_SQL.PROCESS_TSF, or BOL_SQL.PUT_ALLOC, BOL_SQL.PUT_ALLOC_ITEM and BOL_SQL.PROCESS_ALLOC to perform the bulk of the business logic for shipping a transfer or an allocation.

CHECK_ITEMS

This function separates the item details on the message into two groups: one contains sellable items and one contains non-sellable items. The sellable items will be converted into orderable items for shipment.

GET_ORDERABLE_ITEMS

This function builds a collection of orderable items based on the sellable items. Depending on the distro type, it calls ITEM_XFORM_SQL.TSF_ORDERABLE_ITEM_INFO (for transfers) or ITEM_XFORM_SQL.ALLOC_ORDERABLE_ITEM_INFO (for allocations) to distribute the sellable quantities among the orderable items.

HANDLE_ERRORS

This function calls API_LIBRARY.HANDLE_ERRORS to perform error handling.

Filename: bolsqls/b.pls

BOL_SQL.PUT_BOL

This function checks the existence of a shipment based on the BOL number, and creates a shipment if it does not exist.

BOL_SQL.PUT_TSF

This function checks the existence of a transfer in RMS based on the transfer number and does the following:

- If the transfer exists, it updates the transfer to shipped status.
- If the transfer does not exist, it creates a transfer of type 'EG' (externally generated). Since the sending location is already aware of the transfer, the new transfer will **not** be published to the RIB again.

BOL_SQL.PUT_TSF_ITEM

This function checks the existence of an item on a transfer based on the transfer number and the item number. It does the following:

- If the input item is a referential item, fetch and use its transactional level item.
- If the item exists on the transfer, update the quantity buckets on TSFDETAIL.
- If the item does **not** exist on the transfer, create TSFDETAIL. However, new items cannot be added to a closed transfer.
- If sending a pack from a warehouse, reject the message if the sending location does not stock packs, unless the sending location is a finisher.

- For an 'EG' type of transfer to or from a warehouse, a physical warehouse is on the transfer instead of a virtual warehouse. Distribute the transferred quantity to virtual locations based on distribution rules by creating an inventory flow structure and save it on SHIPITEM_INV_FLOW.

BOL_SQL.PROCESS_TSF

This function calls BOL_SQL.SEND_TSF to perform the bulk of the transfer shipment business logic. The key updates performed by this function are:

- If the sending location of the transfer is a finisher, this is the second leg of a multi-legged transfer. Call TSF_WO_COMP_SQL.WO_ITEM_COMP to perform any necessary item transformations, including adjusting inventory and average cost of the old and new items, and writing TRAN_DATA for the adjusted inventory.
- Update inventory (stock_on_hand and tsf_reserved_qty) for the item transferred at the sending location.
- Update inventory (in_transit_qty and tsf_expected_qty) and average cost for the item transferred at the receiving location. Note: average cost is never recalculated for a franchise return at the receiving location, as it is considered a customer return and the average cost of the receiving location is used.
- When the item shipped is a pack item, if the pack item is stocked as a pack at the sending and/or receiving location, inventory is updated for both the pack item (stock_on_hand, tsf_reserved_qty, in_transit_qty, tsf_expected_qty) and the pack component items (pack_comp_soh, pack_comp_resv, pack_comp_intran, pack_comp_exp). On the other hand, if the pack item is **not** stocked as a pack at the sending and/or receiving location, inventory is updated for the component items only (stock_on_hand, tsf_reserved_qty, in_transit_qty, tsf_expected_qty).
- When the item shipped is a simple pack catch weight item, average weight on ITEM_LOC_SOH is updated.
- When the item shipped is a simple pack catch weight item and the pack component's standard UOM is a mass UOM (for example, LBS), the component's inventory is updated by the actual weight shipped.
- Call STKLEDGR_SQL.WRITE_FINANCIALS to write TRAN_DATA records for the sending and receiving locations if the transaction does NOT include a franchise location as the shipping OR receiving location, or if BOTH locations are franchise stores: :
  - **30/32** – for intra-company transfer in/out, in which case the sending and receiving locations belong to the same transfer entity. The transfer is valued at the transfer cost on TSFDETAIL if defined. If not, it is valued at the sending location's WAC. WAC is dependent on the accounting method used, which could be retail accounting or standard cost accounting or average cost accounting. Both WAC and transfer cost are in the sending location's currency.
  - **11/13** – for intra-company markup/markdown. It records the total retail difference between the sending and receiving locations. It is written against either the sending or the receiving location, depending on the settings on the system options (tsf_md_store_to_store_snd_rcv, tsf_md_wh_to_store_snd_rcv, tsf_md_store_to_wh_snd_rcv, tsf_md_wh_to_wh_snd_rcv).
  - **71/72** – for intra-company cost variance. It records the total cost variance as a result of the difference between the sending location's WAC and the transfer cost. It is written against the sending location.
  - **37/38** – for inter-company transfer in/out, in which case the sending and receiving locations belong to different transfer entities. The transfer is valued at

the transfer price on TSFDETAIL. Transfer price is defined in the sending location's currency.

- **17/18** – for inter-company markup/markdown. It records the total retail difference between the transfer price and the sending location's unit retail. It is written against the sending location.
- **65** – for transfer restocking fees if a restocking percentage is defined on the transfer detail. It can be for an inter-company or an intra-company transfer. It is written against the sending locations.
- **28** – for up charges.
  - When a deposit content item is shipped, a TRAN_DATA record is also written for the container item for trans code 30/32 and 37/38. The total cost should be based on the cost of the container.
  - When a simple pack catch weight item is shipped, the total cost is evaluated at the weight shipped. As a result, TRAN_DATA.total_cost reflects the weight shipped for tran codes 37/38, 30/32, 71/72 and 65. However, all the retail calculation is not weight-based. As a result, TRAN_DATA.total_retail and tran codes 17/18, 11/13 do not reflect the actual weight.
  - Call STKLEDGR_SQL.WF_WRITE_FINANCIALS to write TRAN_DATA records for the sending and receiving locations if the transaction is a franchise transaction.

- **20/82** – for franchise order in/out, in which case the sending location is a company location and the receiving location is a franchise store. The transfer is valued at the pricing cost on WF_ORDER_DETAIL (fixed_cost if defined; customer_cost if fixed_cost is not defined). Tran-data 20 is only written if the franchise location is stockholding.
- **24/83** – for franchise return in/out, in which case the sending location is a franchise store and the receiving location is a company location. The transfer is valued at the return unit cost on WF_RETURN_DETAIL. Tran-data 24 is only written if the franchise location is stockholding.
- **84/85** – for franchise markup/markdown. It records the total retail difference between the pricing cost (for franchise orders) or return cost (for franchise returns) and the company location's VAT exclusive unit retail. It is written against the company location.
- **87** – for VAT-in cost, posted in the tran_data.total_cost column against the franchise location:
  - In case of a franchise order, it records the Total Cost in tran_code 20 * Cost VAT Rate at the franchise location.
  - In case of a franchise return, it records the Total Cost in tran_code 24 * Cost VAT Rate at the franchise location, with a negative value for total_cost but positive value for units.
- **88** – for VAT-out retail, posted in the tran_data.total_retail column against the company location:
  - In case of a franchise order, it records the vat-exclusive Total Retail in tran_code 82 * Retail VAT Rate at the company location.
  - In case of a franchise return, it records the vat-exclusive Total Retail in tran_code 83 * Retail VAT Rate at the company location, with a negative value for total_retail but positive value for units.

- **22/23** – for stock adjustment in case of a franchise return with destroy on site. It is only applicable to franchise returns and is written against the company location. If the reason code associated with franchise return destroy on site has a cogs_ind of 'Y', use tran_code 23; otherwise, use tran_code 22.
- **86** – for franchise restocking fees if a restocking percentage is defined on the franchise return detail. It is only applicable to franchise returns and is written against the company location.
- **65** – for franchise restocking fees if a restocking percentage is defined on the franchise return detail. It is only applicable to franchise returns and is written against stockholding franchise locations only.
- **71/72** – for cost variance retail/cost accounting. It records the total cost variance as a result of the difference between the franchise location's WAC and the return unit cost. It is written against the franchise location for franchise returns, if the franchise store is stockholding.
- When a deposit content item is shipped on a franchise transaction, a TRAN_DATA record is also written for the container item. The total cost should be based on the pricing/return cost of the container as defined on wf_order_detail and wf_return_detail.
- Creates shipsku for the item. For a simple pack catch weight item, weight_expected and weight_expected_uom are written along with the qty_expected.
- For a non-franchise transaction, shipsku.unit_retail is the sending location's unit retail. When a break to sell orderable item is shipped, its unit retail is derived from its sellable items. Similarly, in a multi-legged transfer scenario, the sending location can be a finisher. Because a finisher does not have unit retail, the unit retail at the receiving location is used.
- For a franchise order, shipsku.unit_cost contains the sending location's WAC at the time of shipment; shipsku.unit_retail contains the pricing cost. For a franchise return, shipsku.unit_cost is based on the return unit cost; shipsku.unit_retail contains the franchise location's unit retail if it's a stockholding location, or the return unit retail if it is a non-stockholding location.
- For a customer order transfer that is shipped directly to the customer, call STOCK_ORDER_RCV_SQL.TSF_LINE_ITEM to receive the shipment.
- For a franchise transaction, call WF_BOL_SQL.WRITE_WF_BILLING_SALES or WF_BOL_SQL.WF_BILLING_RETURNS to write franchise billing tables.

BOL_SQL.PUT_ALLOC

This function checks the existence of an allocation based on the allocation number, item number and warehouse. If the input item is a referential item, its transactional level item is used. Reject the message if the allocation does not exist.

BOL_SQL.PUT_ALLOC_ITEM

This function checks the existence of allocation detail based on the allocation number and the receiving location. It does the following:

- If the store exists on allocation detail, update the quantity buckets on ALLOC_DETAIL.
- If the store does **not** exist on allocation detail, create ALLOC_DETAIL.
- If any virtual warehouse in the input physical warehouse does not exist on allocation detail, create ALLOC_DETAIL for the primary virtual warehouse.

- If there are multiple virtual warehouses in the same physical warehouse that exist on allocation detail, distribute the transferred quantity to virtual locations based on distribution rules by creating an inventory flow structure.

BOL_SQL.PROCESS_ALLOC

This function calls BOL_SQL.SEND_ALLOC to perform the bulk of the allocation shipment business logic. It does the following:

- Update inventory (stock_on_hand and tsf_reserved_qty) for the item allocated at the sending location.
- Update inventory (in_transit_qty and tsf_expected_qty) and average cost for the item allocated at the receiving location.
- When the item shipped is a pack item, if the pack item is stocked as a pack at the sending/receiving location, inventory is updated for both the pack item (stock_on_hand, tsf_reserved_qty, in_transit_qty, tsf_expected_qty) and the pack component items (pack_comp_soh, pack_comp_resv, pack_comp_intran, pack_comp_exp). On the other hand, if the pack item is **not** stocked as a pack at the sending/receiving location, inventory is updated for the pack component items only (stock_on_hand, tsf_reserved_qty, in_transit_qty, tsf_expected_qty).
- When the item shipped is a simple pack catch weight item, average weight on ITEM_LOC_SOH is updated if the pack is stocked as a pack at the sending/receiving location.
- When the item shipped is a simple pack catch weight item and the pack component's standard UOM is a mass UOM (for example, OZ), component's inventory is updated by the actual weight shipped.
- Call STKLEDGR_SQL.WRITE_FINANCIALS to write TRAN_DATA records for the sending and receiving locations if the transaction does not include NOT a franchise transaction:
  - **37/38** – for inter-company allocation in/out, in which case the sending and receiving locations belong to different transfer entities. Allocations are valued at the sending location's WAC.
  - **30/32** – for intra-company allocation in/out, in which case the sending and receiving locations belong to the same transfer entity. Allocations are valued at the sending location's WAC.
  - **11/13** – for intra-company markup/markdown. It records the total retail difference between the sending and receiving locations. It is written against either the sending or the receiving location, depending on the settings on the system options (tsf_md_store_to_store_snd_rcv, tsf_md_wh_to_store_snd_rcv, tsf_md_store_to_wh_snd_rcv, tsf_md_wh_to_wh_snd_rcv).
  - **28** – for up charges.
  - When a deposit content item is shipped, a TRAN_DATA record is also written for the container item for tran codes 30/32 and 37/38. The total cost should be based on the cost of the container.

    Note: Similar to shipping a transfer, the retail values are not weight-based for a simple pack catch weight item.

- Call STKLEDGR_SQL.WF_WRITE_FINANCIALS to write TRAN_DATA records for the sending and receiving locations if the transaction is a franchise transaction:

> **Note:** Check the PROCESS_TSF for tran-codes posted for a franchise transaction. Since allocation is always from a warehouse, it is only possible to have allocations linked to a franchise order, not a franchise return.

- Creates shipsku for the item. For a simple pack catch weight item, weight_expected and weight_expected_uom are written along with the qty_expected. For an allocation with multiple virtual warehouses in the same physical warehouse on allocation detail, only one shipsku record is written with the qty_expected equal to the ship quantity for the item.
- For an allocation linked to a franchise order, call WF_BOL_SQL.WRITE_WF_BILLING_SALES to write franchise billing tables.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
asnoutcre	ASN Outbound Create Message	ASNOutDesc.xsd

Design Assumptions

- The ASNOut subscription process supports the break to sell functionality. Transfers, allocations and shipments in RMS will only contain break to sell orderable items. Inventory adjustment and stock ledger will be performed on the orderable only, not the sellable.
- The ASNOut subscription process supports the catch weight functionality. It is assumed that a break to sell sellable item cannot be a simple pack catch weight item.
- Catch weight functionality is not completely rounded out in this release. For instance, it is **not** applied to the following areas:
    - Any of the retail calculations (including total_retail on TRAN_DATA and retail markup/markdown);
    - Open to buy buckets;
    - When a catch weight component item's standard UOM is a MASS UOM, TRAN_DATA.units is based on V_PACKSKU_QTY.qty instead of the actual weight.
- An externally generated transfer will contain physical locations. When system options INTERCOMPANY_TSF_IND = 'Y', the stock order receiving process currently does **not** support the receiving of an externally generated transfer that involves a warehouse to warehouse transfer. This is because a physical location does **not** have transfer entities.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
TSFHEAD	Yes	Yes	Yes	No
TSFDETAIL	Yes	Yes	Yes	No

TABLE	SELECT	INSERT	UPDATE	DELETE
TRANSFERS_PUB_INFO	No	Yes	No	No
ALLOC_HEADER	Yes	Yes	Yes	No
ALLOC_DETAIL	Yes	Yes	Yes	No
SHIPMENT	Yes	Yes	Yes	No
SHIPSKU	Yes	Yes	Yes	No
TRAN_DATA	No	Yes	No	No
ITEM_LOC_HIST	No	Yes	No	No
ITEM_LOC_SOH	Yes	Yes	Yes	No
ITEM_LOC	Yes	Yes	No	No
ITEM_ZONE_PRICE	Yes	Yes	No	No
PRICE_HIST	No	Yes	No	No
SHIPITEM_INV_FLOW	No	Yes	No	No
STORE	Yes	No	No	No
WH	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
ITEM_XFORM_HEAD	Yes	No	No	No
ITEM_XFORM_DETAIL	Yes	No	No	No
TSF_XFORM	Yes	No	No	No
TSF_XFORM_DETAIL	Yes	No	Yes	No
TSF_ITEM_COST	Yes	No	Yes	No
TSF_ITEM_WO_COST	Yes	No	No	No
WO_ACTIVITY	Yes	No	No	No
INV_ADJ_REASON	Yes	No	No	No
INV_ADJ	Yes	No	No	No
INV_STATUS_QTY	Yes	Yes	Yes	Yes
DEPS	Yes	No	No	No
CURRENCIES	Yes	No	No	No
CURRENCY_RATES	Yes	No	No	No
PERIOD	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No
WEEK_DATA	Yes	No	No	No
MONTH_DATA	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
ITEM_SUPP_COUNTRY_DIM	Yes	No	No	No
UOM_CLASS	Yes	No	No	No

TABLE	SELECT	INSERT	UPDATE	DELETE
WF_ORDER_HEAD	Yes	No	No	No
WF_ORDER_DETAIL	Yes	No	Yes	No
WF_RETURN_HEAD	Yes	No	No	No
WF_RETURN_DETAIL	Yes	No	Yes	No
WF_BILLING_SALES	No	Yes	No	No
WF_BILLING_RETURNS	No	Yes	No	No

COGS Subscription API

Functional Area

COGS Subscription

Business Overview

The Cost Of Goods Sold (COGS) interface lets a retailer make replacements, which is similar to exchanges. However, replacements involve a different accounting process than exchanges. In a replacement, a retailer replaces a previously purchased item with an equivalent unit. To make this replacement, retailer first places the request and ships the undesirable unit out and later the replacement unit is shipped to the retailer. In RMS, the cost of goods sold interface allows the retailer to make this replacement despite the fact that the exchange is not made simultaneously.

The interface writes the value of the transaction to the transaction data tables. An external system (such as Oracle Retail Data Warehouse) can then extract that data.

The subscription process for COGS adjustment involves an interface which contains item, location, quantity, date, order header media, order line media, and a reason code. These records are inserted into the TRAN_DATA table to affect the stock ledger. Message processing includes a call to STKLEDGER_SQL.TRAN_DATA_INSERT to insert the new transaction to the TRAN_DATA table.

RMS subscribes to integration subsystem COGS messages. This process records the inventory and financial transactions associated with a cost of goods sold message.

Package Impact

Filename: rmssub_cogsb/s.pls

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

CONSUME simply calls different functions within the corresponding VALIDATE and SQL packages.

Before calling any functions, CONSUME narrows I_message down to the specific object being used, depending on the message_type. For example, a 'Cre' or 'Mod' message type usually means a 'Desc' object is being used. A 'Del' message usually means a 'Ref' object is being used. Object narrowing is done using the TREAT function. If the narrowing fails, then the CONSUME function should return an error message to the RIB stating that the object is not valid for this message family.

CONSUME first calls the family's VALIDATE package to validate the contents of the message. The family's SQL package is then called to perform DML.

Business Validation Mode

Filename: rmssub_cogsvalb/s.pls

This function first calls the CHECK_FIELDS function to make sure all required fields are not NULL. Then, the function calls other function as needed to validate all of the information that has been passed to it from the RIB.

DML Module

Filename: rmssub_cogssqlb/s.pls

```
PERSIST
              (O_error_message      IN OUT        RTK_ERRORS.RTK_TEXT%TYPE,
               I_message_type       IN            VARCHAR2,
               I_cogs_rec           IN            RMSSUB_COGS.COGS_REC_TYPE)
```

This function performs the inventory and financial transactions associated with the COGS transaction. The inventory is adjusted at the store location based on the reason code (replacement in/out) provided in the message. In addition a net sale and permanent markdown financial transaction is written to the stock ledger.

Message XSD

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	XML Schema Definition (XSD)
CogsCre	COGS Create Message	CogsDesc.xsd

Design Assumptions

The subscriber makes some assumptions about the publisher's ability to maintain data integrity. The subscriber does not check for duplicate Create messages. It will not check for missing messages because it has no way of knowing what would be missing. It also assumes that messages are sent in the correct sequence.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_LOC	Yes	No	No	No
ITEM_LOC_SOH	No	No	Yes	No
TRAN_DATA	No	Yes	No	No

Cost Change Subscription API

Functional Area

Cost Change

Design Overview

A cost change is performed at the following levels of the organization hierarchy: chain, area, region, district, and store. Unit cost is updated for all stores within the location group. Because warehouses are not part of the organization hierarchy, they are only impacted by cost changes applied at the warehouse level.

The subscription does not create cost change events; it updates the cost of an item in real time. It is intended for use only when RMS is not the system of record for cost changes.

The cost change subscription updates unit costs for item/locations that already exist in RMS. It does not create or delete item/locations in RMS tables.

RMS exposes an API that allows external systems to update unit cost within RMS.

This RMS API subscribes to external cost change modify messages for the purpose of integrating external cost changes maintained in an external system into RMS. It updates unit costs in RMS and writes cost history.

Consume Module

Filename: rmssub_xcostchgs/b.pls

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

This procedure initially ensures that the passed-in message type is a valid type for cost change messages. There is only one valid message type for Cost change messages, XCostchgMod. If the message type is invalid, a status of "E" is 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 is downcast to the actual object using the Oracle treat function. If the downcast fails, a status of "E" is returned to the external system along with an appropriate error message informing the external system that the object passed in is invalid.

If the downcast is successful, then the consume verifies that the message passes all of RMS's business validation by calling the RMSSUB_XCOSTCHG_VALIDATE.CHECK_MESSAGE function. If the message passed RMS business validation, then the function returns true; otherwise, it returns false. If the message has failed RMS business validation, a status of "E" is returned to the external system along with the error message returned from the CHECK_MESSAGE function.

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

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

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

Business Validation Module

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

Filename: rmssub_xcostchgvals/b.pls

```
RMSSUB_XCOSTCHG_VALIDATE.CHECK_MESSAGE
               (O_error_message       IN OUT         VARCHAR2,
                O_cost_change_rec     OUT            COST_CHANGE_REC,
                I_message             IN             RIB_XCostChgDesc,
                I_message_type        IN             VARCHAR2)
```

This function performs all business validation associated with message and builds the cost change record for persistence.

Cost Change Modify

- Checks required fields.
- Verifies supplier's currency.
- Verifies item status.
- If diff IDs are passed in, verifies they are valid for passed in item.
- Verifies item passed in is not a buyer pack.

POPULATING RECORD

- Retrieves the item's transaction level children if the passed-in item is a parent.
- Retrieves all locations based on passed in hierarchy type and value.
- Determines if a location to be updated is the primary location; if so, retrieves the item-supplier-country record to be updated.
- Retrieves all item/location combinations where passed-in supplier/country is the primary supplier/country at an item location.
- Retrieves all orderable buyers pack that the passed-in item, or its children if above transaction level.
- If the recalculate order indicator is 'Y', retrieves all item/locations on approved (and worksheet) orders.
- Populates record with message data.

Package Impact

Filename: rmssub_xcchgsqls/b.pls

```
RMSSUB_XCOSTCHG_SQL.PERSIST
               (O_error_message       IN OUT         VARCHAR2,
                I_dml_rec             IN             COST_CHANGE_RECTYPE ,
                I_message             IN             RIB_XCostChgDesc)
```

Cost Change

- Updates the unit cost on item supplier country location table for all item/locations.
- If one of the locations was a primary location, updates the item supplier country table. Inserts into price history all records for all item/locations related to the supplier/country as the primary supplier/country.
- If average cost method is not used (system option ECL_IND = N), updates the unit cost on item location stock on hand table for all item/locations related to the supplier/country as the primary supplier/country *(***packs do not have cost updated***).*
- If the recalculate order indicator is 'Y', updates all relevant order/item/locations unit cost.
- If pack processing is necessary, repeats the above steps except updating item location stock on hand.

Message XSD

Here are the filenames that correspond with the message type. Please consult the RIB documentation to get a detailed picture of the composition of the message.

Message Type	Message Type Description	XML Schema Definition (XSD)
Xcostchgmod	External Cost Change Modify	XCostChgDesc.xsd

Design Assumptions

- Required fields are shown in the RIB documentation.
- Updating the order cost does not take into account any aspects of building the order cost (estimated landed cost, deals, bracket cost, and so on) and will not work for a base solution.
- This API does not take into account estimated landed cost.
- This API assumes 'A'verage cost accounting. Hence no logic exists for 'S'tandard (last received) cost accounting.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_SUPP_COUNTRY	Yes	No	Yes	No
ITEM_SUPP_COUNTRY_LOC	Yes	No	Yes	No
ITEM_LOC_SOH	Yes	No	Yes	No
STORE	Yes	No	No	No
WH	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
DIFF_GROUP_HEAD	Yes	No	No	No
DIFF_GROUP_DETAIL	Yes	No	No	No
CHAIN	Yes	No	No	No
AREA	Yes	No	No	No
REGION	Yes	No	No	No
DISTRICT	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
ORDLOC	Yes	No	Yes	No
ORDHEAD	Yes	No	No	No
PRICE_HIST	No	Yes	No	No
SYSTEM_OPTIONS	Yes	No	No	No

Currency Exchange Rates Subscription API

Functional Area

Currency Exchange Rates

Business Overview

Currency exchange rates constitute financial information that is published to the Oracle Retail Integration Bus (RIB). A currency exchange rate is the price of one country's currency expressed in another country's currency.

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

Data Flow

An external system will publish a currency exchange rate, thereby placing the currency exchange rate information onto the RIB. RMS will subscribe to the currency exchange rate information as published from the RIB and place the information onto RMS tables depending upon the validity of the records enclosed within the message.

Message Structure

The currency exchange rate message is a flat message that will consist of a currency exchange rate record.

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

Package Impact

Filename: rmssub_curratecres/b.pls

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

Public API Procedures:

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

This procedure accepts a XML file in the form of an Oracle CLOB data type from the RIB (I_message). This message contains a currency exchange rate message consisting of the aforementioned record. The procedure calls the main RMSSUB_CUR_RATES.CONSUME function in order to validate the XML file format and, if successful, parses the values within the file through a series of calls to RIB_XML. The values extracted from the file are then passed on to private internal functions, which validate the values and place them on the currency exchange rate table depending upon the success of the validation.

Private Internal Functions and Procedures (rmssub_curratecre.pls)

Error Handling:

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

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

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

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

Private Internal Functions and Procedures (other):

All of the following functions exist within RMSSUB_CUR_RATES.

Main Consume Function:

```
RMSSUB_CUR_RATES.CONSUME
               (O_error_message      OUT           VARCHAR2,
                I_message            IN            CLOB)
```

This procedure accepts a XML file in the form of an Oracle CLOB data type from the RIB (I_message) from the aforementioned public curratecre procedure whenever a message is made available by the RIB. This message consists of the aforementioned record.

The procedure then validates the XML file format and, if successful, parses the values within the file through a series of calls to RIB_XML. The values extracted from the file are then passed on to private internal functions, which validate the values and place them on the appropriate currency exchange rate database table depending upon the success of the validation.

XML Parsing:

- **PARSE_HEADER** : This function is used to extract the currency exchange rate level information from the currency exchange rate xml file and place that information onto an internal currency exchange rate record.

Validation:

- **PROCESS_HEADER**: After the values are parsed for a particular currency exchange rate record, RMSSUB_CUR_RATES.CONSUME calls this function, which in turn calls various functions inside RMSSUB_CUR_RATES in order to validate the values and place them on the appropriate currency exchange rate table depending upon the success of the validation. CONVERT TYPE is called to validate the passed in currency rate if it exists in the FIF_CURRENCY_XREF table. PROCESS_RATES is called to actually insert or update the currency exchange rate table.
- **CONVERT_TYPE**: This function takes in the current record's exchange rate type and returns the RMS exchange type from the table FIF_CURRENCY_XREF. If no data is found, it should return an error message.
- **PROCESS_RATES**: This function calls VALIDATE_RATES to ensure that the values passed from the message are valid. If all the values are valid, it checks if the currency code exists in the currency exchange rate table. If the currency code does not exist yet, the function INTEREST RATES is called. If not, UPDATE RATES is called.
- **VALIDATE_RATES**: This function passes each value from the record to the function CHECK_NULLS. CHECK_SYSTEM is used for conversion date.
- **CHECK_NULLS**: This function checks if the values passed are NULL. If the passed value is NULL, then an invalid parameter error message is returned.
- **CHECK_SYSTEM**: This function fetches the vdate and the currency code from the period and system options table respectively. If the vdate is greater than the conversion date, an error message is returned. If the passed in currency rate is not the same as the currency rate fetched from the system options table, an error message is returned.

DML Module:

INSERT_RATES: This function inserts into the currency exchange rate table after all of the validations of the values are done.

UPDATE_RATES: This function locks the CURRENCY_RATES table first. After that the table is locked it updates the record in the currency exchange rate table.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
CurrateCre	Currency Rate Create Message	CurrRateDesc.xsd
CurrateCre	Currency Rate Modify Message	CurrRateDesc.xsd

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.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
CURRENCY_RATES	Yes	Yes	Yes	No
SYSTEM_OPTIONS	Yes	No	No	No
PERIOD	Yes	No	No	No
FIF_CURRENCY_XREF	Yes	No	No	No

Diff Group Subscription API

Functional Area

Diff Group

Design Overview

Differentiator subscriptions come into RMS from an external system. With a differentiator group subscription, you create the differentiator group in the external system, and then send that information to RMS. Once the subscription has been received, RMS users can now use the differentiator group that comes from the external system. The group is always sent first; its IDs are sent second.

Differentiators

Differentiators augment RMS' item level structure by allowing you to define more discrete characteristics of an item. You attach differentiators to items to distinguish one item from another. Differentiators (diffs) give you the means to further track merchandise sales transactions. Common types of diffs are size, color, flavor, scent, or pattern.

Diffs consist of:

- **Diff types**; Generic categories of diff IDs such as Size, Color, or Flavor.
- **Diff IDs:** Specific attributes such as black, white, red; small, medium; strawberry, blueberry.
- **Diff groups**; Logical groupings of related diff IDs such as: Women's Pant Sizes, Shirt Colors, or Yogurt Flavors.

This API allows external systems to create, edit, and delete diff groups within RMS. The transaction will be performed immediately upon message receipt so success or failure can be communicated to the calling application.

Diff ID details can be created, edited, or deleted within the diff group message. Diff ID details must be created within a diff group on a diff group create message, they can also be passed in with their own specific message type. Diff ID detail create and modify messages will send a snapshot of the diff group record. Diff ID detail delete messages will be processed separately from the diff group delete because they have their own message types.

Package Impact

Package Impact

Filename: rmssub_xdiffgrps/b.pls

```
RMSSUB_XDIFFGRP.CONSUME
               (O_status_code         IN OUT         VARCHAR2,
                O_error_message       IN OUT         VARCHAR2,
                I_message             IN             RIB_OBJECT,
                I_message_type        IN             VARCHAR2)
```

This procedure will need to initially ensure that the passed in message type is a valid type for diff IDs messages. 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 need to be downcast to the actual object using the Oracle's treat function. 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 calls the RMSSUB_XDIFFGRP_VALIDATE.CHECK_MESSAGE function to determine whether the message is valid. 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. It calls the RMSSUB_XDIFFGRP_SQL.PERSIST_MESSAGE() function. 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_MESSAGE() function.

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

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

Business Validation Module

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

Filename: rmssub_xdiffgrpvals/b.pls

```
RMSSUB_XDIFFGRP_VALIDATE.CHECK_MESSAGE
               (O_error_message      IN OUT        VARCHAR2,
                O_diffgroup_rec      OUT           DIFF_GROUP_REC,
                I_message            IN            RIB_XdiffgrpDesc,
                I_message_type       IN            VARCHAR2)
```

This function performs all business validation associated with the messages and builds the diff group record for persistence.

DIFF GROUP CREATE

- Check required fields.
- Verify diff group ID not used in diff ID table.
- Verify diff ID detail node is populated.
- Verify diff ID details are on diff ID table (not diff group table).
- Populate record with message data.

DIFF GROUP MODIFY

- Check required fields.
- Verify the diff group exists.
- Populate record with message data.

DIFF GROUP DELETE

- Check required fields.
- Verify the Diff group exists.
- Verify diff group is not attached to any items or pack templates.
- Populate record with message data.

DIFF ID CREATE

- Check required fields.
- Verify diff ID detail node is populated.
- Verify diff ID details are on diff ID table (not diff group table).
- Populate record with message data.

DIFF ID MODIFY

- Check required fields.
- Verify diff group exists.
- Verify diff ID detail node is populated.
- Verify diff ID details are on diff ID table (not diff group table).
- Verify diff ID details on diff group detail table.
- Populate record with message data.

DIFF ID DELETE

- Check required fields.
- Verify diff group exists.
- Verify the diff ID exists on diff group table.
- Verify no items or pack templates are using that diff group detail diff ID.
- Populate record with message data.

Bulk or Single DML Module

All insert, update and delete SQL statements are located in the family package. This package is DIFF_GROUP_SQL. The private functions will call this package.

Filename: rmssub_xdiffgrpsqls/b.pls

```
RMSSUB_XDIFFGRP_SQL.PERSIST_MESSAGE
               (O_error_message       IN OUT         VARCHAR2,
                I_diff_group_rec      IN             DIFF_GROUP_REC,
                I_message_type        IN             VARCHAR2,)
```

This function determines what type of database transaction it will call based on the message type.

DIFF GROUP CREATE

- Create messages get added to the Diff group head table.
- Diff group details get added to the diff group detail table.

DIFF GROUP MODIFY

- Modify messages directly update the Diff group head table with changes.

DIFF GROUP DELETE

- \- Delete messages directly remove Diff group head records.

DIFF GROUP DETAIL CREATE

- Create messages get added to the Diff group detail table.

DIFF GROUP DETAIL MODIFY

- Modify messages directly update the Diff group detail table with changes.

DIFF GROUP DETAIL DELETE

- \- Delete messages directly remove Diff group detail records.

Message XSD

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

Message Type	Message Type Description	XML Schema Definition (XSD)
Xdiffgrpdtlcre	Create a diff group detail	XDiffGrpDesc.xsd
Xdiffgrpdtldel	Delete a diff group detail	XDiffGrpRef.xsd
xdiffgrpdtlmod	Modify a diff group detail	XDiffGrpDesc.xsd
xdiffgrpcre	Create a diff group header	XDiffGrpDesc.xsd
xdiffgrpdel	Delete an entire diff group	XDiffGrpRef.xsd
xdiffgrpmod	Modify a diff group header	XDiffGrpDesc.xsd

Design Assumptions

Required fields are shown in the RIB documentation.

Diff IDs and Diff groups must be validated for uniqueness, as they cannot overlap.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
DIFF_IDS	Yes	No	No	No
DIFF_GROUP_HEAD	Yes	Yes	Yes	Yes
DIFF_GROUP_DETAIL	Yes	Yes	Yes	Yes
ITEM_MASTER	Yes	No	No	No
PACK_TMPL_HEAD	Yes	No	No	No
DIFF_RANGE_HEAD	Yes	No	No	No

Diff ID Subscription API

Functional Area

Foundation

Design Overview

The diff ID subscription API provides a means to keep RMS in sync with an external system.

This API allows an external system to create, edit, and delete Diff Ids within RMS. These transactions are performed immediately upon message receipt so success or failure can be communicated to the calling application.

Package Impact

Filename: rmssub_xdiffids/b.pls

```
RMSSUB_XDIFFID.CONSUME
               (O_status_code        IN OUT        VARCHAR2,
                O_error_message      IN OUT        VARCHAR2,
                I_message            IN            RIB_OBJECT,
                I_message_type       IN            VARCHAR2)
```

This procedure initially ensures that the passed in message type is a valid type for diff IDs messages. If the message type is invalid, a status of "E" is 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 is downcast to the actual object using the Oracle treat function. If the downcast fails, a status of "E" is returned to the external system along with an appropriate error message informing the external system that the object passed in is invalid.

If the downcast is successful, then consume verifies that the message passes all of RMS's business validation calling the RMSSUB_XDIFFID_VALIDATE.CHECK_MESSAGE function. If the message passes RMS business validation, then the function returns true; otherwise it returns false. If the message has failed RMS business validation, a status of "E" is returned to the external system along with the error message returned from the CHECK_MESSAGE function.

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

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

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

Business Validation Module

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

Filename: rmssub_xdiffidvals/b.pls

```
RMSSUB_XDIFFID_VALIDATE.CHECK_MESSAGE
               (O_error_message       IN OUT         VARCHAR2,
                O_diffid_rec          OUT            DIFF_ID_REC,
                I_message             IN             RIB_XDiffIDDesc,
                I_message_type        IN             VARCHAR2)
```

This function performs all business validation associated with messages and builds the diff ID record for persistence.

DIFF ID CREATE

- Checks required fields.
- Verifies diff ID not used in diff group head table.
- Populates record with message data.

DIFF ID MODIFY

- Checks required fields.
- Verifies the Diff Id exists.
- Populates record with message data.

DIFF ID DELETE

- Checks required fields.
- Verifies the Diff Id exists.
- Deletes the record with diff ID contained in the message data.

Bulk or single DML module

All insert, update and delete SQL statements are located in the family package. This package is DIFF_ID_SQL. The private functions will call this package.

Filename: rmssub_xdiffidsqls/b.pls

```
RMSSUB_XDIFFID_SQL.PERSIST_MESSAGE
               (O_error_message       IN OUT         VARCHAR2,
                I_diffid_rec          IN             DIFF_ID_REC,
                I_message_type        IN             VARCHAR2,)
```

This function determines what type of database transaction it will call based on the message type.

DIFF ID CREATE

- Create messages get added to the Diff ID table.

DIFF ID MODIFY

- Modify messages directly update the Diff ID table with changes.

DIFF ID DELETE

- Delete messages directly remove Diff ID records.

Message XSD

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

Message Type	Message Type Description	XML Schema Definition (XSD)
xdiffidcre	External Differentiator Create	XDiffIDDesc.xsd
xdiffiddel	External Differentiator Delete	XDiffIDRef.xsd
xdiffidmod	External Differentiator Modify	XDiffIDDesc.xsd

Design Assumptions

Required fields are shown in mapping document.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
DIFF_IDS	Yes	Yes	Yes	Yes
DIFF_GROUP_HEAD	Yes	No	No	No

Direct Ship Receipt Subscription API

Functional Area

Direct Ship Receipt Subscription

Business Overview

In the direct ship receipt process, a retailer does not own inventory, but still records a sale on their books.

An external integration subsystem takes the order and sends it to a supplier.

When an integration subsystem is notified that a direct ship order is sent from the supplier, it publishes a new direct ship (DS) receipt message to the RIB for RMS' subscription purposes. RMS can then account for the data in the stock ledger.

Processing in conjunction with the subscription ensures that the weighted average cost for the item is recalculated.

RMS subscribes to integration subsystem direct ship receipt (DSR) messages. This records the inventory and financial transactions associated with the direct shipment of merchandise.

Package Impact

Filename: rmssub_dsrcpts/b.pls

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

CONSUME simply calls different functions within the corresponding VALIDATE and SQL packages.

Before calling any functions, CONSUME narrows I_message down to the specific object being used, depending on the message_type. For example, a 'Cre' or 'Mod' message type usually means a 'Desc' object is being used. A 'Del' message usually means a 'Ref' object is being used. Object narrowing is done using the TREAT function. If the narrowing fails, then the CONSUME function should return an error message to the RIB stating that the object is not valid for this message family.

CONSUME first calls the family's VALIDATE package to validate the contents of the message. The family's SQL package is then called to perform DML.

Filename: rmssub_dsrcpt_vals/b.pls

```
CHECK_MESSAGE
           (O_error_message   IN OUT         RTK_ERRORS.RTK_TEXT%TYPE,
            O_dsrcpt_rec      OUT NOCOPY     RMSSUB_DSRCPT.DSRCPT_REC_TYPE,
            I_message         IN             "RIB_XOrderDesc_REC",
            I_message_type    IN             VARCHAR2)
```

This function first calls the CHECK_FIELDS function to make sure all required fields are not NULL. Then, the function will call other functions as needed to validate all of the information that has been passed to it from the RIB.

Filename: rmssub_dsrcpt_sqls/b.pls

```
RMSSUB_DSRCPT_SQL.PERSIST
               (O_error_message       IN OUT  RTK_ERRORS.RTK_TEXT%TYPE,
                I_dsrcpt_rec          IN      RMSSUB_DSRCPT.DSRCPT_REC_TYPE,
                I_message_type        IN      VARCHAR2)
```

This function will perform the inventory and financial transactions associated with the direct ship receipt. This includes updating the stock on hand and average cost for the item at the virtual store against which the direct shipment is being received, and, booking the associated purchase to the stock ledger for the item / virtual store.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
Dsrcptcre	Dsrcpt Create Message	DsrcptDesc.xsd

Design Assumptions

The subscriber makes some assumptions with the publisher's ability to maintain data integrity. The subscriber will not check for duplicate create messages. It will not check for missing messages because it has no way of knowing what would be missing. It also assumes that messages are sent in the correct sequence.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_MASTER	Yes	No	No	No
PACKITEM	Yes	No	No	No
ITEM_LOC_SOH	Yes	No	Yes	No
TRAN_DATA	No	Yes	No	No

DSD Deals Subscription API

Functional Area

DSD deals subscription

Business Overview

Direct Store Delivery (DSD) is a delivery of merchandise and/or services to a store without the benefit of a pre-approved purchase order, such as when the supplier drops off merchandise directly in the retail er's store. This process is common in convenience and grocery stores, where suppliers routinely come to restock merchandise. In these cases, the invoice may or may not be given to the store (as opposed to sent to corporate), and the invoice may or may not be paid for out of the register.

RMS subscribes to DSD messages from the RIB. These messages notify RMS of a direct store delivery transaction at a location so that it may record the purchase order and account for it in the store's inventory.

The receipt message that enters RMS includes information such as unit quantity, location, and others. Based on the data, RMS performs the following functionality, as necessary.

- Creates a purchase order.
- Applies any deals
- Creates a shipment
- Receives a shipment.
- Creates an invoice

Note: If ReIM is not running, invoices are not created.

Package Impact

Filename: rmssub_dsddealss/b.pls

```
RMSSUB_DSDDEALS.CONSUME
           (O_status_code            IN OUT        VARCHAR2,
            O_error_message          IN OUT        VARCHAR2,
            I_rib_dsddealsdesc_rec   IN            "RIB_DSDDealsDesc_REC",
            I_message_type           IN            VARCHAR2)
```

This procedure initially ensure that the passed in message type is a valid type for DSD deals. The valid message type for DSD deals messages are listed in a section below.

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

For each header level data in the DSD deals table, call the function COMPLETE_TRANSACTION to persist data to the RMS database.

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

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

RMSSUB_DSDDEALS.COMPLETE_TRANSACTION

This function checks for a shipment record on the shipment table for the DSD being processed. If no shipment record exists, it applies any applicable deals to the DSD order being processed and inserts shipment records into the shipment and shipsku tables for the newly created purchase order. After creating the new shipment, it receives the shipment and approves the order. If the DSD message contains invoice information, it creates the invoice.

RMSSUB_DSDDEALS.HANDLE_ERRORS

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

Message XSD

Here are the filenames that correspond with each message type. Please see *RIB documentation* for each message type in order to get a detailed picture of the composition of each message.

Message Types	Message Type Description	XML Schema Definition (XSD)
dsddealscre	DSD Deals Create Message	DSDDealsDesc.xsd

Design Assumptions

None

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
SHIPMENT	Yes	Yes	No	No
SHIPSKU	No	Yes	No	No
ORDAUTO_TEMP	Yes	No	No	Yes
ORDSKU	Yes	No	No	No
ORDLOC	Yes	No	No	No

DSD Receipt Subscription API

Functional Area

DSD Receipt

Business Overview

Direct store delivery (DSD) is the delivery of merchandise and/or services to a store without the benefit of a pre-approved purchase order. When the delivery occurs, the integration subsystem informs RMS of the receipt so a purchase order is created and it is counted in the store's inventory.

Package Impact

Filename: rmssub_dsds/b.pls

RMSSUB_DSD.CONSUME

```
RMSSUB_DSD.CONSUME
               (O_status_code         IN OUT         VARCHAR2,
                O_error_message       IN OUT         VARCHAR2,
                I_rib_dsddesc_rec     IN             "RIB_DSDReceiptDesc_REC",
                I_message_type        IN             VARCHAR2,
                O_rib_dsddeals_rec    OUT            "RIB_DSDDealsDesc_REC")
```

The passed in message type is validated to ensure it is a valid type for DSD receipts. The valid message type for DSD Receipts messages are listed in a section below.

If the message type is invalid, a status of "E" will 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 DSD_CRE, it performs validation on the values in the message. If the data is valid, it processes the non-merchandise data for delivery costs and detail level data before persisting the data to RMS databases.

If the message type is DSD_MOD, call the GET_ORDER_NO function to find the order number for the DSD.

If the message type is a create message, the O_rib_dsddeals_rec record is populated and passed back to the RIB so that it may be sent to the RMSSUB_DSDDEALS consume function. If the message type is not create, then the O_rib_dsddeals_rec should be set to null.

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

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

RMSSUB_DSD.GET_ORDER_NO

```
GET_ORDER_NO (O_error_message   IN OUT VARCHAR2,
              O_order_no        IN OUT ordhead.order_no%TYPE,
              I_ext_receipt_no  IN     shipment.ext_ref_no_in%TYPE,
              I_store           IN     store.store%TYPE,
              I_supplier        IN     sups.supplier%TYPE)
```

This function is called for message type DSD_MOD. This function retrieves the current order number by searching the shipment tables using the external receipt number, store number and supplier.

RMSSUB_DSD.HANDLE_ERRORS

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

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

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
dsdreceiptcre	DSD Receipt Create Message	DSDReceiptDesc.xsd
dsdreceiptmod	DSD Receipt Modify Message	DSDReceiptDesc.xsd

Design Assumptions

None

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
SHIPMENT	Yes	No	No	No
ORDHEAD	Yes	No	No	No

Freight Terms Subscription API

Functional Area

Foundation

Business Overview

Freight terms are financial arrangement information that is published to the Oracle Retail Integration Bus (RIB) from a financial system. Freight terms are the terms for shipping (for example, the freight terms could be a certain percentage of the total cost; a flat fee per order, etc).. RMS subscribes to freight terms messages held on the RIB. After confirming the validity of the records enclosed within the message, the RMS database is updated with the information.

Required fields in the message include a unique freight terms ID and a description.

Message Structure

The freight term message is a flat message that will consist of a freight term record.

Package Impact

Filename: rmssub_frttermcres/b.pls
rmssub_fterms/b.pls

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

Public API Procedures

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

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

Private Internal Functions and Procedures (rmssub_frttermcre.pls):

Error Handling

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

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

All error handling in the internal RMSSUB_FTERM package and all errors that occur during subscription in the RMSSUB_FRTTERMCRE package (and whatever packages it calls) will flow through this function.

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

Private Internal Functions and Procedures (rmssub_fterm.pls):

All of the following functions exist within RMSSUB_FTERM.

Main Consume Function

```
RMSSUB_FTERM.CONSUME
              (O_status_code         IN OUT         VARCHAR2,
               O_error_message       IN OUT         VARCHAR2,
               I_message_clob        IN             CLOB)
```

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

The procedure then validates the XML file format and, if successful, parses the values within the file through a series of calls to RIB_XML. The values extracted from the file will then be passed on to private internal functions, which will validate the values and place them on the appropriate freight term database table depending upon the success of the validation.

XML Parsing

PARSE_FTERM

This function will used to extract the freight term level information from the Freight Term XML file and place that information onto an internal freight term record.

Validation

PROCESS_FTERM

After the values are parsed for a particular freight term record, RMSSUB_FTERM.CONSUME will call this function, which will in turn call various functions inside RMSSUB_FTERM in order to validate the values and place them on the appropriate FREIGHT_TERMS table depending upon the success of the validation.

Message XSD

Below are the filenames that correspond with each message type. Please consult the RIB documentation for each message type in order to get a detailed picture of the composition of each message.

Message Types	Message Type Description	XML Schema Definition (XSD)
FrtTermCre	Freight Term Create Message	FrtTermDesc.xsd

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.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
FREIGHT_TERMS	Yes	Yes	Yes	No

GL Chart of Accounts Subscription API

Functional Area

GL Chart of Accounts

Business Overview

Before RMS publishes stock ledger data to an external financial application, it must receive that application's General Ledger Chart Of Accounts (GLCOA) structure. RMS accomplishes this through a subscription process.

A chart of account is essentially the financial application's debit and credit account segments (for example, company, cost center, account, and others) that applies to RMS product hierarchy. In some financial applications, this is known as Code Combination IDs (CCID). On receiving the GLCOA message data, RMS populates the data to the FIF_GL_ACCT table. The GL cross-reference form is used to associate the appropriate department, class, subclass, and location data to a CCID that allows the population of that data to the GL_FIF_CROSS_REF table.

An external system publishes GL Chart of Accounts, thereby placing the GL chart of accounts information to RIB (Retek Information Bus). RMS subscribes the GL chart of accounts information as published from the RIB and places the information in RMS tables depending upon the validity of the records enclosed within the message.

Package Impact

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

Public API Procedures:

Filename: rmssub_glcoacreb.pls

```
RMSSUB_ GLCOACRE.CONSUME
               (O_status_code         IN OUT         VARCHAR2,
                O_error_message       IN OUT         VARCHAR2,
                I_message             IN             CLOB)
```

This procedure accepts a XML file in the form of an Oracle CLOB data type from the RIB (I_message). This message contains a GL chart of accounts message consisting of the aforementioned record. The procedure places a call to the main RMSSUB_GLCACCT.CONSUME function in order to validate the XML file format and, if successful, parse the values within the file through a series of calls to RIB_XML. The values extracted from the file is passed to private internal functions, which validates the values and place them on the GL chart of accounts table depending upon the success of the validation.

Private Internal Functions and Procedures (rmssub_glcoacreb.pls):

Error Handling:

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

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

All error handling in the internal RMSSUB_GLCACCT package and all errors that occur during subscription in the RMSSUB_GLCOACRE package (and whatever packages it calls) flows through this function.

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

Private Internal Functions and Procedures (other):

Filename: rmssub_glcacctb.pls

Main Consume Function:

```
RMSSUB_GLCACCT.CONSUME
               (O_ERROR_MESSAGE      OUT     VARCHAR2,
                I_MESSAGE            IN      CLOB)
```

This procedure accepts a XML file in the form of an Oracle CLOB data type from the RIB (I_message) from the public rmssub_glcoacre.consume procedure whenever a message is available in RIB. This message consists of the aforementioned record.

The procedure validates the XML file format and if successful, parses the values within the file through a series of calls to RIB_XML. The values extracted from the file is passed to a private internal functions, which validates the value and places to a appropriate GL chart of accounts database table depending upon the success of the validation.

XML Parsing:

```
PARSE_HEADER
            (O_ERROR_MESSAGE   OUT            VARCHAR2,
             O_GLACCT_RECORD   OUT            GLACCT_RECTYPE,
             I_GLACCT_ROOT     IN OUT         xmldom.DOMElement)
```

This function extracts the GL chart of accounts level information from the GL Chart of Accounts XML file and places the information to an internal GL Chart of Accounts record.

Record is based upon the record type glacct_rectype.

Validation:

PROCESS_HEADER

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

PROCESS_GLACCT

Function PROCESS_GLACCT takes the input GL record and places the information to a local GL record which is used in the package to manipulate the data. It calls a series of support functions to perform all business logic on the record.

INSERT_GLACCT

Function INSERT_GLACCT inserts any valid account on the GL table. It is called from PROCESS_GLACCT.

UPDATE_GLACCT

Function UPDATE_GLACCT updates any valid account on the GL table. It is called from PROCESS_GLACCT.

VALIDATE_GLACCT

Function VALIDATE_GLACCT is a wrapper function which is used to call CHECK_NULLS, CHECK_ATTRS for any GL record input into the package.

CHECK_NULLS

Function CHECK_NULLS checks an input value if it is null. If so, an error message is created based on the passed in record type.

CHECK_ATTRS

Function CHK_ATTRS is called within the validation function of this package to ensure that RMS will not accept incomplete data from a financial interface when sent through RIB. This function checks to ensure that each description that is input also has an attribute that it describes.

Message XSD

The GL chart of accounts message is a flat message consists of a GL chart of accounts record.

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

Here are the filenames that correspond with each message type. Please consult the RIB documentation for each message type to get detailed information of the composition of each message.

Message Types	Message Type Description	XML Schema Definition (XSD)
Glcoacre	Glco Create Message	GLCOADesc.xsd

Design Assumptions

Required fields are shown in the RIB documentation.

Many ordering functionalities that are available on-line are not supported through this API. Triggers related to these functionalities must be turned off.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
FIF_GL_ACCT	Yes	Yes	Yes	No

Inventory Adjustment Subscription

Functional Area

Inventory Adjustment

Business Overview

RMS receives requests for inventory adjustments from an integration subsystem through the inventory adjustment subscription. The requests contain information about the item, the stockholding location, the quantity, the specific disposition change, and the reason for the adjustment. RMS uses data in these requests to:

- Adjust overall quantities of stock on hand for an item at a location
- Adjust the availability of item-location quantities. For unavailable inventory adjustments, all quantity adjustment goes to the non-sellable bucket.

After initial processing from the integration subsystem RMS performs the following tasks:

- Validates the item-location combinations and adjustment reasons
- Updates stock on hand data for the item at the location
- Inserts stock adjustment transaction codes on the RMS stock ledger
- Adjusts quantities by inventory status for item/location combination
- Create an audit trail for the inventory adjustment by item, location, inventory status and reason

Inventory Quantity and Status Evaluation

RMS evaluates inventory adjustments to decide if overall item-location quantities have changed, or if the statuses of quantities have changed.

The FROM_DISPOSITION and TO_DISPOSITION tags in the message are evaluated to determine if there is a change in overall quantities of an item at a location. For the given item and quantity reported in the message, if either tag contains a null value, RMS evaluates that as a change in overall quantity in inventory.

In addition, if the message shows a change to the status of existing inventory, RMS evaluates this to determine if that change makes a quantity of an item unavailable.

Stock Adjustment Transaction Codes

Whenever the status or quantity of stock changes, RMS writes transaction codes to adjust inventory values in the stock ledger. The two types of inventory adjustment transaction codes are:

- Adjustments to total stock on hand, where positive and negative adjustments are made to total stock on hand. In this case, a 'Stock Adjustment' transaction (TRAN_CODE = '22' or '23' if the cost of goods indicator associated with the inventory adjustment reason code is 'Y') is inserted on the Stock Ledger (TRAN_DATA table) for both the retail and cost value of the adjustment
- Adjustments to unavailable (non-sellable) inventory. In this case, an 'Unavailable Inventory Transfer' transaction (TRAN_CODE = '25') is inserted on the Stock Ledger (TRAN_DATA table).

L10N Localization Decoupling Layer:

This is a layer of code which enables decoupling of localization logic that is only required for certain country-specific configuration. This layer affects the RIB API flows including Inventory Adjustment subscription. This allows RMS to be installed without requiring customers to install or use this localization functionality, where not required.

Package Impact

Filename: rmssub_invadjusts/b.pls

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

This procedure will initially ensure that the passed in message type is a valid type for inventory adjustment messages. The valid message type for an inventory adjustment message is listed in a section below.

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

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

RMSSUB_INVADJUST.CONSUME_INVADJ

- (O_status_code IN OUT VARCHAR2,
- O_error_message IN OUT VARCHAR2,
- I_message IN RIB_OBJECT,
- I_message_type IN VARCHAR2,
- I_check_l10n_ind IN VARCHAR2)
- Perform localization check. If localized, invoke localization logic through L10N_SQL decoupling layer for procedure key 'CONSUME_INVADJ'. If not localized, call CONSUME_INVADJ for normal processing.

RMSSUB_INVADJUST.CONSUME_INVADJ

- (O_error_message IN OUT VARCHAR2,
- IO_L10N_RIB_REC IN OUT L10N_OBJ)
- Public function to call RMSSUB_INVADJUST.CONSUME_INVADJUST_CORE.

RMSSUB_INVADJUST.CONSUME_INVADJ_CORE

- (O_error_message IN OUT VARCHAR2,
- I_message IN RIB_OBJECT,
- I_message_type IN VARCHAR2)
- This function contains the main processing logic.

If the downcast is successful, then consume needs to verify that the message passes all of RMS's business validation. It calls the INVADJ_SQL function to perform validation and to insert or update records in the database when the message is valid. If the message passed RMS business validation and is successfully persisted in the database then a successful status is returned to the CONSUME. If the message fails RMS business validation or encounters any other errors, a status of "E" is returned to the external system along with the error message.

```
RMSSUB_INVADJUST.PROCESS_INVADJ
              (O_error_message      IN OUT        VARCHAR2,
               I_message            IN            "RIB_InvAdjustDesc_REC")
```

This function calls CHECK_ITEMS, an internal function that checks for any sellable only "break to sell" items and separates these items into an object table for further processing. A table of the corresponding orderable items and quantities for the sellable items is built to submit to the inventory adjustment process. INVADJ_SQL.PROCESS_INVADJ is called for the table of regular items and the table of "break to sell" items to perform all business validation and desired functionality associated with an inventory adjustment message.

Filename: invadjs/b.pls

INVADJ_SQL.BUILD_PROCESS_INVADJ

This function performs business validation and desired functionality for an inventory adjustment message. It includes the following:

- Check required fields: item, location, adj_qty, user_id, adj_date.
- Verify that the to_disposition or from_disposition or both fields are populated. Both cannot be NULL.
- Verify that an orderable but non-sellable and non-inventory item cannot be an inventory adjustment item.

- If the item is a simple pack catch weight item, verify that weight and weight UOM are either both defined or both NULL, and, if populated, that the weight UOM is in the MASS UOM class.
- Verify that the item is a tran-level or a reference item. When a reference item is passed in, its parent item's inventory is adjusted.
- Verify that the item/loc relation exists and create it if it does not exist.
- If adjusting a pack at a warehouse, receive_as_type must be 'P' (pack) on ITEM_LOC.
- Verify that from disposition and to disposition are valid inventory status codes (on INV_STATUS_CODES).
- If the location is a warehouse, then physical location is on the message. The adjusted quantity is distributed among the virtual locations of the physical location.
- For available stock on hand, the items are added to the update records for updating the ITEM_LOC_SOH table and a tran code 22 or 23 is prepared for writing the TRAN_DATA records. For external finisher location type and for transformable orderable items, the unit_retail is a calculated value, based on package calls for these two exception cases.
- If cost of goods indicator of the inventory adjustment reason code is 'Y', use tran_code 23 instead of 22.
- For unavailable stock on hand, the unavailable quantities are computed before the items or the pack components are added to the update records for updating the ITEM_LOC_SOH table and a tran code 25 data is prepared for writing the TRAN_DATA records. For external finisher location type and for transformable orderable items, the unit_retail is calculated with the appropriate package call for these two exception cases.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
invadjustcre	Inventory Adjustment Create Message	InvAdjustDesc.xsd

Design Assumptions

None

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_LOC_SOH	Yes	Yes	Yes	No
TRAN_DATA(VIEW)	No	Yes	No	No
INV_ADJ	No	Yes	No	No
INV_STATUS_QTY	No	Yes	Yes	Yes
INV_ADJ_REASON	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
INV_STATUS_CODES	Yes	No	No	No
TSFHEAD	Yes	No	No	No
SHIPSKU	Yes	No	No	No

Inventory Request Subscription API

Functional Area

Inventory Request Subscription

Business Overview

RMS receives requests for inventory from an integration subsystem through the inventory request subscription.

Store ordering allows for all items to be ordered by the store and fulfilled by an RMS process. RMS fulfills a store's request regardless of replenishment review cycles, delivery dates, and any other factors that may restrict a request from being fulfilled. However, delivery cannot always be guaranteed on or before the store requested due date, due to supplier or warehouse lead times and other supply chain factors that may restrict on-time delivery.

Store ordering can be used to request inventory for any items that are on the 'Store Order' type of replenishment. The store order replenishment process requires the store to request a quantity and builds the recommended order quantity (ROQ) based on the store's requests. Requests for store order items that will not be reviewed prior to the date requested by the store are fulfilled through a one-off process (executed real-time through the API) that creates warehouse transfers and/or purchase orders to fulfill the requested quantities.

This API can also be used for items setup on other types of replenishment. In this case the store requested quantities will be added 'above and beyond' the calculated recommended order quantities. This API can also be used for items not setup on auto-replenishment. In this case the one-off process described above will be used to create a PO or transfer utilizing attributes defined for the item/location.

Package Impact

Filename: rmssub_invreqs/b.pls

```
RMSSUB_INVREQ.CONSUME
               (O_status_code         IN OUT         VARCHAR2,
                O_error_message       IN OUT         VARCHAR2,
                I_message             IN             RIB_OBJECT,
                I_message_type        IN             VARCHAR2,
                O_rib_error_tbl       OUT            RIB_ERROR_TBL)
```

This procedure initially downcasts the generic RIB_OBJECT to the actual object using the Oracle treat function.

If the downcast is successful, it will empty out the cache of inserts and updates to the store_orders table and to the PL/SQL ITEM_TBL table. This is done by calling INV_REQUEST_SQL.INIT function. Global variables to be used are initialized by the function RMSSUB_INVREQ_ERROR.INIT. This is called before processing any item/store order request.

Input from the header level info is then validated. If any of the required header level info is NULL, the entire request is rejected; however, there is no need to write to the error table.

Once the header level info has passed validation, RMSSUB_INVREQ_ERROR.BEGIN_INVREQ is called to hold the header level values into global variables which may be used to build an error record when necessary. Each item is processed by calling INV_REQUEST_SQL.PROCESS.

The cache for the STORE_ORDERS table and the PL/SQL ITEM_TBL table is populated by calling INV_REQUEST_SQL.FLUSH function. At the end of the inventory request process, the RMSSUB_INVREQ_ERROR.FINISH function is called to pass a copy of the global error table (if any error exists) which is sent to the RIB for further processing.

Filename: rmssub_invreq_errors/b.pls

Most of the functions included are called by the RMSSUB_INVREQ.CONSUME procedure to process inventory requests.

```
RMSSUB_INVREQ_ERROR.INIT
               (O_error_message       IN OUT         RTK_ERRORS.RTK_TEXT%TYPE,
                I_message_type        IN             VARCHAR2)
```

This function initializes all of the global variables which include the RIB_OBJECTS that are used to process the inventory request.

```
RMSSUB_INVREQ_ERROR.BEGIN_INVREQ
               (O_error_message       IN OUT         RTK_ERRORS.RTK_TEXT%TYPE,
                I_request_id          IN             NUMBER,
                I_store               IN             STORE_ORDERS.STORE%TYPE,
                I_request_type        IN             VARCHAR2)
```

This function populates the global variables using the header level values to create an error record whenever necessary.

```
RMSSUB_INVREQ_ERROR.ADD_ERROR
               (O_error_message       IN OUT         RTK_ERRORS.RTK_TEXT%TYPE,
                I_error_desc          IN             VARCHAR2,
                I_error_object        IN             RIB_OBJECT)
```

This function is called whenever an error is encountered during the processing of the inventory request. It adds the error type/description and error object on the global error table.

RMSSUB_INVREQ_ERROR.FINISH

This function is called after processing the inventory request. It passes out a copy of the global error table (if any error is present) to the RIB for further processing.

RMSSUB_INVREQ_ERROR.GET_MESSAGE_KEY

This function gets the key from a SQL_LIB error message. If the error message is just text without any parameters, the entire message is passed back out as the key.

Filename: invrequests/b.pls

```
INV_REQUEST_SQL.PROCESS
           (O_error_message   IN OUT         VARCHAR2,
            I_store           IN             STORE_ORDERS.STORE%TYPE,
            I_request_type    IN             VARCHAR2,
            I_item            IN             STORE_ORDERS.ITEM%TYPE,
            I_need_qty        IN             STORE_ORDERS.NEED_QTY%TYPE,
            I_uop             IN             UOM_CLASS.UOM%TYPE,
            I_need_date       IN             STORE_ORDERS.NEED_DATE%TYPE)
```

This function does all the validation and processing of the inventory request. It creates a record for STORE_ORDERS or LP_ITEM_TBL (PL/SQL table for adhoc requests).

INV_REQUEST_SQL.VERIFY_REPL_INFO (local)

This function retrieves the replenishment information. If the request type is 'IR' and the item is not set up on replenishment, set adhoc to 'Y'. Item requests with request type of 'SO' or NULL must have store order replenishment set up in RMS for that item. The need date must be after the next replenishment delivery date if the store order has been rejected by replenishment. If the need date is before the next replenishment review date for both request types, set adhoc to 'Y'.

INV_REQUEST_SQL.FUNCTION CONVERT_NEED_QTY (local)

This function converts the need quantity to 'E'aches for Packs.

INV_REQUEST_SQL.PREPARE_AD_HOC (local)

This function is called if the Adhoc indicator is set to 'Y'. It writes the request to the PL/SQL table that will be passed to the function call CREATE_ORD_TSF_SQL.CREATE_ORD_TSF to create an order or transfer.

INV_REQUEST_SQL.VERIFY_ON_STORE (local)

This function checks to see if the item request already exists on STORE_ORDER. If it exists, call PREPARE_UPDATE to update the need quantity to include the new need quantity. If it does not, call PREPARE_INSERT to insert into STORE_ORDER table.

INV_REQUEST_SQL. PREPARE_INSERT (local)

This function checks the PL/SQL table that contains the BULK INSERT records. If a record exists on the PL/SQL table, update the qty.

INV_REQUEST_SQL. PREPARE_UPDATE (local)

This function adds a record to the PL/SQL table that contains the BULK UPDATE records.

INV_REQUEST_SQL. FLUSH (local)

This function does the actual insert or update to STORE_ORDERS.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
InvReqCre	Inventory Request Create Message	InvReqDesc.xsd

Design Assumptions

- RMS will round quantities using the store order multiple when an order is created for a warehouse.
- Up charges will always be applied to a transfer when they can be defaulted.
- RMS will validate that all items belong to the same department when department level ordering (supplier) or department level transfers (warehouse) are being used.
- RMS will validate that an item is not a consignment item if the order is for a warehouse.
- RMS will validate that a store is open when the store is being transferred to.
- This API supports non-fatal error processing. If an error is encountered in one inventory request detail, it will log and return the error to the RIB via RIB_ERROR_TBL and continue processing the next detail.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
STORE_ORDERS	Yes	Yes	Yes	No
REPL_ITEM_LOC	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
SUPS	Yes	No	No	No
ITEM_LOC_SOH	Yes	No	No	No
TSFHEAD	No	Yes	Yes	No
TSFDETAIL	Yes	Yes	Yes	No
ORDHEAD	No	No	Yes	No

Item Subscription API

Functional Area

Item

Design Overview

When this API accepts messages with create message types, it inserts the data into the following staging tables:

- SVC_ITEM_MASTER
- SVC_PACKITEM (in the case of a pack)
- SVC_ITEM_SUPPLIER
- SVC_ITEM_COUNTRY
- SVC_ITEM_SUPP_COUNTRY
- SVC_ITEM_SUPP_COUNTRY_DIM
- SVC_ITEM_SUPP_MANU_COUNTRY
- SVC_UDA_ITEM_LOV
- SVC_UDA_ITEM_FF
- SVC_UDA_ITEM_DATE

The SVC_VAT_ITEM table is populated with data is defaulted from the item's department. Optionally, the records can be inserted into the SVC_VAT_ITEM table to override these defaults. The messages with modify message types consist of snapshots of records for updating the ITEM_MASTER, ITEM_SUPPLIER, ITEM_SUPP_COUNTRY, ITEM_SUPP_COUNTRY_DIM, and ITEM_SUPP_MANU_COUNTRY tables after being processed from the corresponding staging tables.

Item messages include the required detail nodes for the supplier and supplier/country. If the item is not a non-sellable pack, the item/zone/price node is also required. Optional nodes can be included in the message for supplier/country, pack components, and item/VAT relationships.

Items must be created and maintained following a logical hierarchy as outlined by the referential integrity of the item database tables: Item parents before child items; item components before items that are packs; items before item-suppliers; item/suppliers before item/supplier/countries; items before item/locations (a separate API), and so on. If the items are not created and maintained in the preceding tables then the message fails.

The Create and Modify messages are hierarchical with required detail nodes of suppliers and supplier/countries and optional nodes for price zones, supplier/country, and VAT codes. If the item is a pack item, the pack component node is required. In the header modify message, the detail nodes are not populated, but the full header node is sent.

The detail level Create or Modify messages contains the item header record and one to many detail records in the node or nodes. For example, the message type of XItemSupMod could have one or more supplier details to update in the ITEM_SUPPLIER table.

The modify messages contain a snapshot of the record for update rather than only the fields to be changed. The auto-creation of item children using differentiator records attached to an item parent, as currently occur using RMS online processes, is not supported in this API.

The delete messages contain only the primary key field for the item, supplier, supplier/country or VAT/item record that is to be deleted. When a delete message is processed, the item is not immediately deleted but is added to the daily purge table. Deleting the item is a batch process.

A major functionality added to RMS is the support of Brazil Localization. This introduced a layer of code to enable decoupling of localization logic that is only required for country-specific configuration. This layer affects the RIB API flows including Xitem subscription.

L10N Localization Decoupling Layer:

ORFM introduced in the 13.2 release as a **bolt-on** product to RMS to handle Brazil-specific fiscal management. Even though ORFM and RMS exist in the same database schema and ORFM cannot be installed separately without RMS, we must ensure that RMS is decoupled from ORFM. So that non-Brazilian clients can install RMS without ORFM. To achieve that, an **L10N decoupling layer** is introduced.

In the context of XItem subscription API, when RMS consumes an XItem message from an external system, if the message involves a localized country, the message must be routed to a 3rd party application (For example, Mastersaf) to calculate tax and/or to ORFM for setting up fiscal item attributes. In that case, RMS's XItem subscription API (rmssub_xitem and related packages) will call Mastersaf and/or ORFM through an L10N decoupling layer.

Import Brazil-specific Fiscal Item Attributes to the Flex Attributes Extension Table (ITEM_COUNTRY_L10N_EXT):

XItem API supports importing of Brazil fiscal item attributes to RMS through the 'xitemctrycre' (create item country) messsage type. Client will need to populate the **"RIB_BrXItemCtryDesc_TBL"** node in the XItemDesc message family. The XItem API writes data to the **ITEM_COUNTRY_L10N_EXT** table based on the meta-data definition of the 'ITEM_COUNTRY' entity.

The structure of the XItemDesc message family is the following:

Here the Brazilian fiscal item attributes are populated.

"RIB_XItemDesc_REC"
-- XItemCtryDesc_TBL "RIB_XItemCtryDesc_TBL"
-- LocOfXItemCtryDesc_TBL "RIB_LocOfXItemCtryDesc_TBL"
-- BrXItemCtryDesc_TBL "RIB_BrXItemCtryDesc_TBL"

Supported fiscal item attributes include:

- SERVICE_IND
- ORIGIN_CODE
- CLASSIFICATION_ID
- NCM_CHAR_CODE
- EX_IPI
- PAUTA_CODE
- SERVICE_CODE
- FEDERAL_SERVICE
- STATE_OF_MANUFACTURE
- PHARMA_LIST_TYPE

When the message is persisted to the database, if the message type is 'xitemctrycre' (for example, create Item Country), then the above Brazilian fiscal item attributes are imported to the corresponding extension table of ITEM_COUNTRY_L10N_EXT through an L10N localization layer.

Package Impact

Consume Module

Filename: rmssub_items/b.pls

```
RMSSUB_XITEM.CONSUME (O_status_code       IN OUT VARCHAR2,
O_error_message  IN OUT VARCHAR2,
I_message              IN         RIB_OBJECT,
I_message_type      IN         VARCHAR2)
```

This procedure needs to initially to ensure that the passed in message type is a valid type for organizational hierarchy messages. The valid message types for organizational hierarchy messages are listed in following sections:

- If the message type is invalid, a status of "E" should be returned to the external system along with an appropriate error message informing the external system that the status is invalid.
- If the message type is valid, the generic RIB_OBJECT needs to be downcast to the actual object using the Oracle's treat function. There will be an object type that corresponds with each message type. If the downcast fails, a status of "E" is returned to the external system along with an appropriate error message informing the external system that the object passed in is invalid.
- If the downcast is successful, then consume calls the RMSSUB_XITEM_POP_RECORD.POPULATE function to populate all the fields in the item collections. It is then persisted to the RMS database via RMSSUB_XITEM_SQL.PERSIST function where contents of the collections are inserted into the staging tables. If the database persistence fails, the function returns false. A status of "E" is returned to the external system along with the error message returned from the PERSIST function.

Once the message has been successfully persisted, no other procedure is required to be processed. A success status, "S", is returned to the external system indicating that the message has been successfully received and persisted to the staging tables in the RMS database. Records in the staging tables are processed by the CORESVC_ITEM.PROCESS function where the validations occur. After having passed the data level validations, the items will be inserted into the main RMS tables, ITEM_MASTER, ITEM_COUNTRY, VAT_ITEM, ITEM_SUPPLIER, ITEM_SUPP_COUNTRY, ITEM_SUPP_COUNTRY_DIM, ITEM_SUPP_MANU_COUNTRY, PACKITEM, ITEM_ZONE_PRICE, UDA_ITEM_FF, UDA_ITEM_DATE, UDA_ITEM_LOV.

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

Bulk or Single DML Module

All Insert, Update and Delete SQL statements are located in the family packages. The private functions call these packages.

Filename: rmssub_xitem_sql

```
RMSSUB_XITEM_SQL.PERSIST
               (O_error_message       IN OUT         VARCHAR2,
                I_message_type        IN             VARCHAR2,
                I_message             IN             RIB_XItemDesc,
                I_item_rec            IN             RMSSUB_ITEM.ITEM_API_REC)
```

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

ITEM CREATE

- Inserts a record in the SVC_ITEM_MASTER table
- Calls all the "insert" functions to insert records into the following tables:
  - SVC_ITEM_COUNTRY
  - SVC_ITEM_SUPPLIER
  - SVC_ITEM_SUPP_COUNTRY
  - SVC_ITEM_SUPP_MANU_COUNTRY
  - SVC_PACKITEM (optional)
  - SVC_VAT_ITEM (optional)
  - SVC_UDA_ITEM_FF(optional)
  - SVC_UDA_ITEM_LOV(optional)
  - SVC_UDA_ITEM_DATE(optional)

ITEM MODIFY

- Inserts a record in SVC_ITEM_MASTER. It will be used to update the ITEM_MASTER table.

ITEM DELETE

- Inserts a record in the SVC_ITEM_MASTER. The record will be processed and inserted into the DAILY_PURGE table.

ITEM COUNTRY CREATE

- Inserts records in SVC_ITEM_COUNTRY. It will be used to update the ITEM_COUNTRY table.
- For Brazil, the records in SVC_ITEM_COUNTRY will be used to update the ITEM_COUNTRY_L10N_EXT table through L10N decoupling layer (L10N_FLEX_API_SQL.PERSIST_L10N_ATTRIB)

ITEM_COUNTRY DELETE

- Inserts record in the SVC_ITEM_COUNTRY table. This will be used to delete records in the ITEM_COUNTRY table and ITEM_COUNTRY_L10N_EXT table.

ITEM_SUPPLIER CREATE

- Inserts records in the SVC_ITEM_SUPPLIER table. This will be used to insert records in ITEM_SUPPLIER.

ITEM_SUPPLIER MODIFY

- Inserts records in the SVC_ITEM_SUPPLIER table. This will be used to modify the ITEM_SUPPLIER table.

ITEM_SUPPLIER DELETE

- Inserts records in the SVC_ITEM_SUPPLIER table for item. This will be used to delete from the ITEM_SUPPLIER table.

ITEM_SUPP_COUNTRY CREATE

- Inserts records in SVC_ITEM_SUPP_COUNTRY. This will be used to insert into the ITEM_SUPP_COUNTRY tabl.e

ITEM_SUPP_COUNTRY MODIFY

- Inserts records in the SVC_ITEM_SUPP_COUNTRY table. This will be used to update the ITEM_SUPP_COUNTRY table.

ITEM_SUPP_COUNTRY DELETE

- Inserts records in the SVC_ITEM_SUPP_COUNTRY table. This will be used to delete records from the ITEM_SUPP_COUNTRY table.

ITEM_SUPP_MANU_COUNTRY CREATE

- Inserts records in the SVC_ITEM_SUPP_MANU_COUNTRY table. This will be used to insert into the ITEM_SUPP_MANU_COUNTRY table.

ITEM_SUPP_MANU_COUNTRY MODIFY

- Inserts records in the SVC_ITEM_SUPP_MANU_COUNTRY table. This will be used to update the ITEM_SUPP_MANU_COUNTRY table.

ITEM_SUPP_MANU_COUNTRY DELETE

- Inserts records in the SVC_ITEM_SUPP_MANU_COUNTRY table. This will be used to delete from the ITEM_SUPP_MANU_COUNTRY table.

ITEM_SUPP_COUNTRY_DIM CREATE

- Inserts records in the SVC_ITEM_SUPP_COUNTRY_DIM table. This will be used to insert into the ITEM_SUPP_COUNTRY_DIM table.

ITEM_SUPP_COUNTRY_DIM MODIFY

- Inserts records in the SVC_ITEM_SUPP_COUNTRY_DIM table. This will be used to update the ITEM_SUPP_COUNTRY_DIM table.

ITEM_SUPP_COUNTRY_DIM DELETE

- Inserts records in the SVC_ITEM_SUPP_COUNTRY_DIM table. This will be used to delete records from the ITEM_SUPP_COUNTRY_DIM table.

PACKITEM CREATE

- Inserts records in the SVC_PACKITEM table. Records from the staging table will be used to insert into PACKITEM and SVC_PACKITEM AND update ITEM_SUPP_COUNTRY_LOC and/or ITEM_SUPP_COUNTRY with calculated unit_cost.

VAT_ITEM CREATE

- Inserts records in the SVC_VAT_ITEM table. The records are inserted into VAT_ITEM or replace any default records that were created from department/VAT.

VAT_ITEM DELETE

- Inserts records in the SVC_VAT_ITEM table. The records will be used to delete from VAT_ITEM.

ITEM_UDA CREATE

- Inserts records into the SVC_UDA_ITEM_DATE, SVC_UDA_ITEM_LOV and SVC_UDA_ITEM_FF tables. The records will then be inserted into the corresponding RMS base tables.

ITEM_UDA MODIFY

- Inserts records into the SVC_UDA_ITEM_DATE, SVC_UDA_ITEM_LOV and SVC_UDA_ITEM_FF tables. The records will then be used to update records in the corresponding RMS base tables.

ITEM_UDA DELETE

- Inserts records into the SVC_UDA_ITEM_DATE, SVC_UDA_ITEM_LOV and SVC_UDA_ITEM_FF tables. The records will then be used to update records from the corresponding RMS base tables.

Message XSD

Here are the filenames that correspond with each message type. Refer the *Oracle Retail Integration Bus documentation set* for each message type for details on the composition of each message.

Message Types	Message Type Description	XML Schema Definition (XSD)
XItemCre	Item Create Message	XItemDesc.xsd
XItemMod	Item Modify Message	XItemDesc.xsd
XItemDel	Item Delete Message	XItemRef.xsd
XItemSupCre	Item/Supplier Create Message	XItemDesc.xsd
XItemSupMod	Item/Supplier Modify Message	XItemDesc.xsd
XItemSupDel	Item/Supplier Delete Message	XItemRef.xsd
XItemSupCtyCre	Item/Supplier/Country Create Message	XItemDesc.xsd
XItemSupCtyMod	Item/Supplier/Country Modify Message	XItemDesc.xsd
XItemSupCtyDel	Item/Supplier/Country Delete Message	XItemRef.xsd
XISCMfrCre	Item/Supplier/Country of Manufacture Create Message	XItemDesc.xsd
XISCMfrMod	Item/Supplier/ Country of Manufacture Modify Message	XItemDesc.xsd
XISCMfrDel	Item/Supplier/ Country of Manufacture Delete Message	XItemRef.xsd

Message Types	Message Type Description	XML Schema Definition (XSD)
XISCDimCre	Item/Supplier/Country/Dimension Create Message	XItemDesc.xsd
XISCDimMod	Item/Supplier/Country/Dimension Modify Message	XItemDesc.xsd
XISCDimDel	Item/Supplier/Country/Dimension Delete Message	XItemRef.xsd
XItemVatCre	Item/Vat Create Message	XItemDesc.xsd
XItemVatDel	Item/Vat Delete Message	XItemRef.xsd
XitemCtryCre	Item/Country Create Message	XItemCtryDesc.xsd
XitemCtryDel	Item/Country Delete Message	XItemCtryRef.xsd
XitemUdaCre	Item/UDA Create Message	XItemDesc.xsd
XitemUdaDel	Item/UDA Delete Message	XItemRef.xsd

Design Assumptions

Item/Supplier/Country/Location relationships are not handled by this API.

Item/location relationships are not handled by this API; they are handled in a separate Item Location Subscription API.

Tables

RPM is called to set the initial pricing for the item. This populates tables in the RPM system.

TABLE	SELECT	INSERT	UPDATE	DELETE
SVC_ITEM_MASTER	Yes	Yes	Yes	No
SVC_ITEM_SUPPLIER	Yes	Yes	Yes	Yes
SVC_ITEM_SUPP_COUNTRY	Yes	Yes	Yes	Yes
SVC_ITEM_SUPP_MANU_COUNTRY	Yes	Yes	Yes	Yes
SVC_ITEM_SUPP_COUNTRY_DIM	Yes	Yes	Yes	Yes
SVC_PACKITEM	No	Yes	No	No
SVC_VAT_ITEM	Yes	Yes	Yes	Yes
SYSTEM_OPTIONS	Yes	No	No	No
SVC_ITEM_COUNTRY	Yes	Yes	No	Yes
SVC_UDA_ITEM_DATE	Yes	Yes	Yes	Yes
SVC_UDA_ITEM_FF	Yes	Yes	Yes	Yes
SVC_UDA_ITEM_LOV	Yes	Yes	Yes	Yes
SVC_PROCESS_TRACKER	Yes	Yes	No	No

Item Location Subscription API

Functional Area

Items – Locations

Design Overview

Item locations can be maintained at the following levels of the organization hierarchy: chain, area, region, district, and store. Records are maintained for all stores within the location group. Because warehouses are not part of the organization hierarchy, they are only impacted by records maintained at the warehouse level. If building item-locations by organizational hierarchy, only locations in the hierarchy that do not already exist on item-location will be built.

Item locations can only be created for a single item. However, levels of the organization hierarchy are available for maintenance in order to facilitate location-level processing into RMS. The detail node is required for both create and modify messages.

Item supplier country locations will be created for the passed-in primary supplier/country if they do not already exist. If primary supplier/country locations are not passed in, then they will default from the item's primary supplier/country and a location will be created, if it does not already exist.

Item locations are required to be interfaced into RMS in active status. There is no delete function in this API. Instead, item locations can be put into inactive, discontinued, or deleted status. However, they will be deleted if the associated item is purged. If building item-locations by store or warehouse, then each passed-in location must not already exist as an item-location.

A major functionality added to RMS is the support of Brazil Localization. This introduced a layer of code to enable decoupling of localization logic that is only required for country-specific configuration. This layer affects the RIB API flows including XItemLoc subscription.

L10N Localization Decoupling Layer:

RFM is introduced in the 13.2 release as a **bolt-on** product to RMS to handle Brazil-specific fiscal management. Even though RFM and RMS exist in the same database schema and RFM cannot be installed separately without RMS, we must ensure that RMS is decoupled from RFM. This is so that non-Brazilian clients can install RMS without RFM. To achieve that, an L10N decoupling layer is introduced.

In the context of XITEMLOC subscription API, when RMS consumes an XITEMLOC message from an external system, the message must be routed to a 3rd party tax application (for example, Mastersaf) for tax calculation if the message involves ranging an item to a new Brazilian location. In that case, RMS's XItemLoc subscription API (rmssub_xitemloc and related packages) will call Mastersaf through an L10N de-coupling layer.

Package Impact

Consume Module

Filename: rmssub_xitemlocs/b.pls

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

This procedure needs to initially ensure that the passed in message type is a valid type for item location messages. If the message type is invalid, a status of "E" should be returned to the external system along with an appropriate error message informing the external system that the status is invalid.

If the message type is valid, the generic RIB_OBJECT needs to be downcast to the actual object using the Oracle's treat function. 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 calls the RMSSUB_XITEMLOC_VALIDATE.CHECK_MESSAGE function to determine whether the message is valid. 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. It calls the RMSSUB_XITEMLOC_SQL.PERSIST_MESSAGE() function. If the database persistence fails, the function returns false. A status of "E" should be returned to the external system along with the error message returned from the PERSIST_MESSAGE() 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_XITEMLOC.HANDLE_ERROR() – This is the standard error handling function that wraps the API_LIBRARY.HANDLE_ERROR function.

Also detail RIB object RIB_XItemLocDtl_REC is modified to support Store serialization.

Business Validation Module

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

Filename: rmssub_xitemlocvals/b.pls

```
RMSSUB_XITEMLOC_VALIDATE.CHECK_MESSAGE
               (O_error_message       IN OUT         VARCHAR2,
                O_ITEMLOC_rec         OUT            ITEMLOC_REC,
                I_message             IN             RIB_XItemLocDesc,
                I_message_type        IN             VARCHAR2)
```

This function performs all business validation associated with message and builds the item locations record for persistence.

ITEMLOC CREATE

- Check required fields.
- Verify primary supplier/country exists on Item-supplier-country.
- If creating locations by store or warehouse, verify passed in locations do not currently exist.
- If item is a buyer pack, verify receive as type is valid based on item's order as type.
- Default required fields not provided (store order multiple, taxable indicator, local item description, primary supplier/country, receive as type).
- Build item-location records.
- Build price history records.

ITEMLOC MODIFY

- Check required fields
- Populate item-location record.

Package Impact

Filename: rmssub_xitemlocsqls/b.pls

```
RMSSUB_XITEMLOC_SQL.PERSIST
               (O_error_message       IN OUT         VARCHAR2,
                I_dml_rec             IN             ITEMLOC_RECTYPE ,
                I_message             IN             RIB_XITEMLOCDesc)
```

ITEMLOC CREATE

- Insert a record into the item-location table.
- Insert a record into the item-location-stock on hand table.
- If necessary, insert a record into the item supplier country location table.
- Insert a record into the price history table.

ITEMLOC MODIFY

- Update item-location table.

Message XSD

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

Message Type	Message Type Description	XML Schema Definition (XSD)
xitemloccre	External item locations create	XItemlocDesc.xsd
xitemlocMod	External item locations odification	XItemlocDesc.xsd

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_SUPP_COUNTRY	Yes	No	No	No
ITEM_SUPP_COUNTRY_LOC	Yes	Yes	No	No
ITEM_LOC_SOH	Yes	Yes	Yes	No
STORE	Yes	No	No	No
WH	Yes	No	No	No
ITEM_LOC	Yes	Yes	Yes	No
SYSTEM_OPTIONS	Yes	No	No	No
PRICE_HIST	No	Yes	No	No
ITEM_MASTER	Yes	No	No	No
POS_MODS	No	Yes	No	No
PACKITEM_BREAKOUT	Yes	No	No	No
CHAIN	Yes	No	No	No
AREA	Yes	No	No	No
REGION	Yes	No	No	No
DISTRICT	Yes	No	No	No
PACKITEM	Yes	No	No	No
RPM_ITEM_ZONE_PRICE	Yes	No	No	No
CURRENCIES	Yes	No	No	No
ELC_TABLES	Yes	No	No	No
VAT_ITEM	Yes	No	No	No
PARTNER	Yes	No	No	No

Item Reclassification Subscription API

Functional Area

Items – Reclassification

Design Overview

RMS subscribes to item reclassification messages that are published by an external system. This subscription is necessary in order to keep RMS in sync with the external system. The retailer can view the pending reclassifications online in RMS.

This API allows external systems to create and delete item reclassification events within RMS.

At least one detail must be passed for a valid reclassification message. Reclassification items can be created or deleted within the reclassification message. Reclass item creates will send a snapshot of the reclass event. However, reclass item deletes do not require any header information as items are unique for reclassification and items may be deleted across reclass events.

Only level one items can be interfaced via this API. If the item is a pack, only non-simple packs can be interfaced. Simple pack items will be reclassified when their component is reclassified.

During the reclassification batch process, it will determine if any pack items exist in RMS that contain the items or any of that item's children being reclassified. If such a pack exists and contains no other items, the batch process adds the pack to the reclassification event being created in RMS.

It is valid for a reclassification event to be created for a department/class/subclass not yet existing but planning to exist. This is valid as long as they department/class/subclass is scheduled to be created on or prior to the reclassification taking effect.

Deleting reclassifications can either occur by:

- Items on a reclass event or across events.
- A single reclassification event.
- All reclassification events on a particular event date (deletion through the use of the reclass_date may result in the deletion of numerous reclass events).
- All reclassification events.

Deleting a reclassification header will require either a reclass no, reclass date, or purge all ind.

Bulk or Single DML Module

Consume Module

Filename: rmssub_xitemrclss/b.pls

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

This procedure needs to initially ensure that the passed in message type is a valid type for item reclassification messages. If the message type is invalid, a status of "E" should be returned to the external system along with an appropriate error message informing the external system that the status is invalid.

If the message type is valid, the generic RIB_OBJECT needs to be downcast to the actual object using the Oracle's treat function. 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 calls the RMSSUB_XITEMRCLS_VALIDATE.CHECK_MESSAGE function to determine whether the message is valid. 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. It calls the RMSSUB_XITEMRCLS_SQL.PERSIST_MESSAGE() function. If the database persistence fails, the function returns false. A status of "E" should be returned to the external system along with the error message returned from the PERSIST_MESSAGE() 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_XITEMRCLS.HANDLE_ERROR() – This is the standard error handling function that wraps the API_LIBRARY.HANDLE_ERROR function.

Business Validation Module

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

Package Impact

Filename: rmssub_xitemrclsvals/b.pls

```
RMSSUB_XITEMRCLS_VALIDATE.CHECK_MESSAGE
               (O_error_message      IN OUT         VARCHAR2,
                O_ITEMRCLS_rec       OUT            ITEMRCLS_REC,
                I_message            IN             RIB_XITEMRCLSDesc,
                I_message_type       IN             VARCHAR2)
```

This function performs all business validation associated with message and builds the item reclassification record for persistence.

ITEMRCLS CREATE

- Check required fields
- Verify items not on existing reclassification
- Validate the reclassification date (must be today or greater).
- Verify hierarchy of item being reclassified to (either an existing hierarchy or a pending hierarchy that will be created prior to the item reclassification)
- Verify non-consignment related reclassification and no unit and dollar stocks performed on items
- Build reclassification records

ITEMRCLS DELETE

- Check required fields
- For reclassification header deletes, verify deleting by either reclassification number, reclassification (event) date, or purging all reclassifications
- Populate record

ITEMRCLS DETAIL CREATE

- Check required fields
- Verify items not on existing reclassification
- Validate the reclassification date (must be today or greater).
- Verify hierarchy of item being reclassified to (either an existing hierarchy or a pending hierarchy that will be created prior to the item reclassification)
- Verify non-consignment related reclassification and no unit and dollar stocks performed on items
- Build reclassification records

ITEMRCLS DETAIL DELETE

- Check required fields
- Populate record.

Package Impact

Filename: rmssub_xitemrclssqls/b.pls

```
RMSSUB_XITEMRCLS_SQL.PERSIST
               (O_error_message       IN OUT         VARCHAR2,
                I_dml_rec             IN             ITEMRCLS_RECTYPE ,
                I_message             IN             RIB_XITEMRCLSDesc)
```

ITEMRCLS CREATE

- Insert a record into the reclass header table
- Insert a record into the reclass item table

ITEMRCLS DETAIL DELETE

- Delete from the reclass item table.

ITEMRCLS DELETE

- If purging all records, delete all from reclass item table.
- If purging all records, delete all from reclass header table.
- If not purging, delete from reclass item for reclass number or all reclass for an event date.
- If not purging, delete from reclass header for reclass number or all reclass for an event date.

ITEMRCLS DELETE

- Delete from reclass item for all items on record.
- If no items exist for an event, delete the reclass event.

Message XSD

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

Message Type	Message Type Description	XML Schema Definition (XSD)
xitemrclscre	External item reclassification create	XItemRclsDesc.xsd
xitemrclsdtlcre	External item reclassification detail create	XItemRclsDesc.xsd
Xitemrclsdel	External item reclassification delete	XitemRclsRef.xsd
Xitemrclsdtldel	External item reclassification detail delete	XItemRclsRef.xsd

Design Assumptions

Orderable buyer packs as 'E'aches will not be allowed to be reclassified if department level ordering is Y in RMS.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
RECLASS_HEAD	Yes	Yes	No	Yes
RECLASS_ITEM	Yes	Yes	No	Yes
ITEM_MASTER	Yes	No	No	No
PACKITEM	Yes	No	No	No
PACKITEM_BREAKOUT	Yes	No	No	No
V_MERCH_HIER	Yes	No	No	No

Location Trait Subscription API

Functional Area

Location Trait

Design Overview

The Location Trait Subscription API processes incoming data from an external system to create, edit and delete location traits in RMS. This data is processed immediately upon message receipt so success or failure can be communicated to the external application.

Package Impact

Consume Module

Filename: rmssub_xloctrts/b.pls

```
RMSSUB_XLOCTRT.CONSUME
               (O_status_code         IN OUT         VARCHAR2,
                O_error_message       IN OUT         VARCHAR2,
                I_message             IN             RIB_OBJECT,
                I_message_type        IN             VARCHAR2)
```

This procedure will need to initially ensure that the passed in message type is a valid type for loc traits messages. 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 need to be downcast to the actual object using the Oracle's treat function. 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 calls the RMSSUB_XLOCTRT_VALIDATE.CHECK_MESSAGE function to determine whether the message is valid. 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. It calls the RMSSUB_XLOCTRT_SQL.PERSIST_MESSAGE() function. 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_MESSAGE() function.

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

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

Business Validation Module

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

Filename: rmssub_xloctrtvals/b.pls

```
RMSSUB_XLOCTRT_VALIDATE.CHECK_MESSAGE
               (O_error_message      IN OUT         VARCHAR2,
                O_loctrait_rec       OUT            LOC_TRAITS_REC,
                I_message            IN             RIB_XLocTraitDesc,
                I_message_type       IN             VARCHAR2)
```

This function performs all business validation associated with messages and builds the location trait record for persistence.

LOCATION TRAIT CREATE

- Check required fields.
- Populate record with message data.

LOCATION TRAIT MODIFY

- Check required fields.
- Verify the location trait exists.
- Populate record with message data.

LOCATION TRAIT DELETE

- Check required fields.
- Verify the location trait exists.
- Populate record with message data.

Bulk or Single DML Module

All insert, update and delete SQL statements are located in the family package. This package is LOC_TRAITS_SQL. The private functions will call this package.

Filename: rmssub_xloctrtsqls/b.pls

```
RMSSUB_XLOCTRT_SQL.PERSIST_MESSAGE
               (O_error_message      IN OUT         VARCHAR2,
               I_loc_trait_rec       IN             LOC_TRAIT_REC,
               I_message_type        IN             VARCHAR2,)
```

This function determines what type of database transaction it will call based on the message type.

LOCATION TRAIT CREATE

- Create messages get added to the location trait table.

LOCATION TRAIT MODIFY

- Modify messages directly update the location trait table with changes.

LOCATION TRAIT DELETE

- Delete messages directly remove location trait records.

Message XSD

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

Message Type	Message Type Description	XML Schema Definition (XSD)
xloctrtcre	External Location Trait Create	XLocTrtDesc.xsd
xloctrtdel	External Location Trait Delete	XLocTrtRef.xsd
xloctrtmod	External Location Trait Modification	XLocTrtDesc.xsd

Design Assumptions

Required fields are shown in RIB documentation.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
LOC_TRAITS	Yes	Yes	Yes	Yes

Merchandise Hierarchy Subscription API

Functional Area

Foundation Data

Business Overview

The merchandise hierarchy allows the retailer to create the relationships that are necessary to support the product management structure of a company. This hierarchy reflects a classification of merchandise into multi-level descriptive categorizations to facilitate the planning, tracking, reporting, and management of merchandise within the company.

If RMS is not the system of record for merchandise hierarchy information for an implementation, then this API may be used to create, update or delete elements of the merchandise hierarchy, including division, group, department, class, and subclass, based on an external system.

Division and group deletes also occur immediately upon receipt of the message. However, departments, classes, and subclasses will not actually be deleted from the system upon receipt of the message. Instead, they will be added to the DAILY_PURGE table, where validation will occur to ensure the records can be deleted. For more on this

batch process, see the *Retail Merchandising System Operations Guide, Volume 1 - Batch Overviews and Designs*.

Department VAT records can be created and edited within the department message (VAT records are not deleted). VAT creates can be passed in with a department create message, or they can be passed in with their own specific message type. VAT region and VAT codes records must exist prior to creating department VAT records. Also, when passing in a new VAT region to an existing department with attached items, the VAT information will default to all items.

The merchandise hierarchy must be created from the highest level down. Conversely, the hierarchy must be deleted from the lowest level up. Each lower level references a parent level. This means a department is associated with a group; a class is associated with a department; and a subclass is associated with department/class combination because classes are not unique across departments.

Package Impact

Filename: rmssub_xmrchhrs/b.pls

```
RMSSUB_XMRCHHR.CONSUME
               (O_status_code         IN OUT         VARCHAR2,
                O_error_message       IN OUT         VARCHAR2,
                I_message             IN             RIB_OBJECT,
                I_message_type        IN             VARCHAR2)
```

This procedure will call the appropriate merchandise hierarchy family package based on the message type passed in.

- Any company message type will call RMSSUB_XMRCHHRCOMP.CONSUME
- Any division message type will call RMSSUB_XMRCHHRDIV.CONSUME
- Any group message type will call RMSSUB_XMRCHHRGRP.CONSUME
- Any department message type will call RMSSUB_XMRCHHRDEPT.CONSUME
- Any class message type will call RMSSUB_XMRCHHRSCLS.CONSUME
- Any subclass message type will call RMSSUB_XMRCHHRCLS.CONSUME

Filename: rmssub_xmrchhr[family_name]vals/b.pls

```
RMSSUB_XMRCHHR[family_name]_VALIDATE.CHECK_MESSAGE
       (O_error_message       IN OUT         VARCHAR2,
        O_[family_name]_rec   OUT            NOCOPY  MERCH_SQL.[FAMILY_NAME]_TYPE,
        I_message             IN             RIB_XMrchHr[family_name]Desc,
        I_message_type        IN             VARCHAR2)
```

This function performs all business validation associated with messages and builds the merchandise hierarchy record for persistence. It should be noted that some of the business validation is referential or involves uniqueness. This validation is handled automatically by the referential integrity constraints and the unique indexes implemented on the database.

Filename: rmssub_xmrchhr[family_name]sqls/b.pls

```
RMSSUB_XMRCHHR[family_name]__SQL.PERSIST_MESSAGE
               (O_error_message       IN OUT         VARCHAR2,
                I_[family_name]_rec   IN MERCH_SQL.[FAMILY_NAME]_TYPE,
                I_message_type        IN             VARCHAR2,)
```

All insert, update and delete SQL statements are located in the family package. This package is MERCH_SQL. The private functions will call this package. This function determines what type of database transaction it will call based on the message type.

Message XSD

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

Message Type	Message Type Description	XML Schema Definition (XSD)
xmrchhrclscre	External Create Class	XMrchHrClsDesc.xsd
xmrchhrcompcre	External Create Company	XMrchHrCompDesc.xsd
xmrchhrdeptcre	External Create Department	XMrchHrDeptDesc.xsd
xmrchhrdivcre	External Create Division	XMrchHrDivDesc.xsd
xmrchhrgrpcre	External Create Group	XMrchHrGrpDesc.xsd
xmrchhrsclscre	External Create Subclass	XMrchHrSclsDesc.xsd
xmrchhrclsdel	External Delete Class	XMrchHrClsRef.xsd
xmrchhrdeptdel	External Delete Department	XMrchHrDeptRef.xsd
xmrchhrdivdel	External Delete Division	XMrchHrDivRef.xsd
xmrchhrgrpdel	External Delete Group	XMrchHrGrpRef.xsd
xmrchhrsclsdel	External Delete Subclass	XMrchHrSclsRef.xsd
xmrchhrvatcre	External Merch Hierarchy VAT create	XMrchHrDeptDesc.xsd
xmrchhrvatmod	External Merch Hierarchy VAT modify	XMrchHrDeptDesc.xsd
xmrchhrclsmod	External Modify Class	XMrchHrClsDesc.xsd
xmrchhrcompmod	External Modify Company	XMrchHrCompDesc.xsd
xmrchhrdeptmod	External Modify Department	XMrchHrDeptDesc.xsd
xmrchhrdivmod	External Modify Division	XMrchHrDivDesc.xsd
xmrchhrgrpmod	External Modify Group	XMrchHrGrpDesc.xsd
xmrchhrsclsmod	External Modify Subclass	XMrchHrSclsDesc.xsd

Design Assumptions

- A department cannot be set up as both direct cost and consignment. Either the budget markup percent or the budget intake percent must be passed in. If RPM is installed, the average tolerance percent and maximum average counter must be greater than zero.

Table Impact

Note that this section does **not** include the tables checked in the Daily Purge batch process.

TABLE	SELECT	INSERT	UPDATE	DELETE
COMPHEAD	Yes	Yes	Yes	No
DIVISION	Yes	Yes	Yes	Yes
DAILY_PURGE	No	Yes	No	No

TABLE	SELECT	INSERT	UPDATE	DELETE
GROUPS	Yes	Yes	Yes	Yes
DEPS	Yes	Yes	Yes	No
VAT_DEPS	Yes	Yes	Yes	No
CLASS	Yes	Yes	Yes	No
SUBCLASS	Yes	Yes	Yes	No

Merchandise Hierarchy Reclassification Subscription API

Functional Area

Merchandise Hierarchy Reclassification

Business Overview

RMS can subscribe to merchandise hierarchy reclassification messages that are published by an external system for retailers who manage their hierarchies in a system outside RMS. This API allows for pending merchandise hierarchy reclassification events to be created, modified or deleted. A separate batch process will read the information off the pending merchandise hierarchy table and create or modify the merchandise hierarchy information in RMS once the change effective date arrives. This API does not accept messages to delete an existing merchandise hierarchy. Any deletion should be done through the Merchandise Hierarchy Subscription API instead. Furthermore, this API will not allow moving a class or subclass between departments. In RMS, a new class and/or subclass needs to be created and the items moved as part of an item reclassification and then the old class and/or subclass deleted.

Package Impact

Consume Module

Rmssub_xmrchhrclss/b.pls

```
RMSSUB_XMRCHHRRCLS.CONSUME
               (O_status_code        IN OUT        VARCHAR2,
                O_error_message      IN OUT        VARCHAR2,
                I_message            IN            RIB_OBJECT,
                I_message_type       IN            VARCHAR2)
```

This procedure will initially ensure that the passed in message type is a valid type for merchandise hierarchy reclassification messages. If the message type is invalid, a status of 'E' – Error will 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 be downcast to the actual object using the Oracle's Treat function. If the downcast fails, a status of 'E' will 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 will verify that the message passes all of RMS's business validation. If the message has failed RMS business validation, a status of 'E' will 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. If the database persistence fails, the function will return false. A status of 'E' will be returned to the external system along with the error message returned from the PERSIST_MESSAGE() function.

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

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

Business Validation Module

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

Filename: rmssub_xmrchhrrclsvals/b.pls

```
RMSSUB_XMRCHHRRCLS_VALIDATE.CHECK_MESSAGE
       (O_error_message       IN OUT         VARCHAR2,
        O_pend_merch_hier_rec        OUT            PEND_MERCH_HIER%ROWTYPE,
            I_message                IN             "RIB_XMrchHrRclsDesc_REC",
            I_message_type           IN             VARCHAR2)
```

This function performs all business validation associated with the messages and builds the merchandise hierarchy record for persistence.

CREATE

- Check required fields. Required fields vary based on hierarchy level.

  Adding New Hierarchy

  - Verify passed in hierarchy does not already exist.
  - Verify parent hierarchy already exists on merchandise hierarchy or pending merchandise hierarchy tables.

  Modifying Existing Hierarchy

  - Verify passed in hierarchy already exists.
  - Verify that class and subclass hierarchies have passed in parent hierarchy in an existing hierarchy (i.e. classes and subclasses are not allowed to be reclassified into another department).
- Populate record with message data

MODIFY

- Check required fields.
- Verify the hierarchy is already pending.
- Populate record with message data.

DELETE

- Check required fields.
- Verify a pending hierarchy event exists.
- Verify no pending hierarchy events exist for levels below the passed in hierarchy level.
- Populate record with message data.

Bulk or single DML module

All insert, update and delete SQL statements are located in the family package. This package is MERCH_RECLASS_SQL. The private functions will call this package.

Filename: rmssub_ xmrchhrrclssqls /b.pls

```
RMSSUB_XMRCHHRRCLS_SQL.PERSIST_MESSAGE
        (O_error_message            IN OUT        VARCHAR2,
        I_pend_merch_hier_rec       IN            PEND_MERCH_HIER%ROWTYPE,
        I_message_type              IN            VARCHAR2)
```

This function determines what type of database transaction it will call based on the message type.

CREATE

- Create messages get added to the pending merchandise hierarchy table.

MODIFY

- Modify messages directly update the pending merchandise hierarchy table with changes.

DELETE

- Delete messages get removed from the pending merchandise hierarchy table.

Message XSD

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

Message Type	Message Type Description	XML Schema Definition (XSD)
xmrchhrrclscre	Create Merchandise Hierarchy Reclassification	XMrchHrRclsDesc.xsd
xmrchhrrclsdel	Delete Merchandise Hierarchy Reclassification	XMrchHrRclsRef.xsd
xmrchhrrclsmod	Modify Merchandise Hierarchy Reclassification	XMrchHrRclsDesc.xsd

Design Assumptions

NA

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
DIVISION	Yes	No	No	No
GROUPS	Yes	No	No	No
DEPS	Yes	No	No	No
CLASS	Yes	No	No	No
SUBCLASS	Yes	No	No	No
PEND_MERCH_HIER	Yes	Yes	Yes	Yes

Organizational Hierarchy Subscription API

Functional Area

Foundation Data

Business Overview

If RMS is not the system of records for organizational hierarchy information for an implementation, then this API may be used to create, update or delete elements of the hierarchy, including chain, area, region, and district, based on an external system. The organization hierarchy subscription also assigns existing location traits to or deletes them from elements of the organization hierarchy. Although stores are part of the organization hierarchy, they differ sufficiently to require their own subscription API.

RMS exposes an API that allows external systems to create, edit, and delete chain, area, region, and districts. All creates, updates, and deletes will occur immediately upon receipt of the message.

Location trait records are created and deleted within the area, region, and district messages. Location trait creates is passed in with the area, region, or district create message, or they can be passed in with their own specific create message type attached to the aforementioned messages. The location trait creates a message that sends a snapshot of the hierarchy record they are attached to. A location trait delete message is processed separately from the hierarchy delete messages.

The organizational hierarchy must be created from the highest level down. Conversely, the hierarchy must be deleted from the lowest level up. Each lower level references a parent level. This means an area is associated with a chain, a region is associated with an area, and a district is associated with a region. Location traits are removed from a hierarchy before it can be removed.

Package Impact

Filename: rmssub_xorghiers/b.pls

```
RMSSUB_XORGHIER.CONSUME
               (O_status_code        IN OUT        VARCHAR2,
                O_error_message      IN OUT        VARCHAR2,
                I_message            IN            RIB_OBJECT,
                I_message_type       IN            VARCHAR2)
```

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

If the message type is valid, the generic RIB_OBJECT will 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 is successful, then consume will verify that the message passes all of RMS's business validation. It calls the RMSSUB_XORGHIER_VALIDATE.CHECK_MESSAGE function to determine whether the message is valid. Once the message has passed RMS business validation, it is persisted to the RMS database. Once the message has been successfully persisted, a success status, "S", is returned to the external system indicating that the message has been successfully received and persisted to the RMS database.

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

Filename rmssub_xohvals/b.pls

```
RMSSUB_XORGHIER_VALIDATE.CHECK_MESSAGE
       (O_error_message   IN OUT     VARCHAR2,
        O_org_hier_rec    OUT        NOCOPY  ORGANIZATION_SQL.ORG_HIER_REC,
        I_message         IN         RIB_XOrgHrDesc,
        I_message_type    IN         VARCHAR2)
```

This function performs all business validation associated with messages and builds the organizational hierarchy record for persistence.

Filename: rmssub_xorghr_sqls/b.pls

```
RMSSUB_XORGHR_SQL.PERSIST_MESSAGE
           (O_error_message   IN OUT         VARCHAR2,
            I_hier_level      IN             VARCHAR2,
            I_org_hier_rec    IN             ORGANIZATIONAL_SQL.ORG-HIER_REC,
            I_message_type    IN             VARCHAR2,)
```

This function determines what type of database transaction it will call based on the message type. All insert, update and delete SQL statements are located in the family package. This package is ORGANIZATIONAL_SQL. The private functions will call this package.

Message XSD

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

Message Type	Message Type Description	XML Schema Definition (XSD)
XOrgHrCre	External Create Organizational Hierarchy	XOrgHrDesc.xsd
XOrgHrLocTrtCre	External Create Organizational Hierarchy Location Trait	XOrgHrDesc.xsd
XOrgHrDel	External Delete Organizational Hierarchy	XOrgHrRef.xsd
XOrgHrLocTrtDel	External Delete Organizational Hierarchy Location Trait	XOrgHrRef.xsd
XOrgHrMod	External Modify Organizational Hierarchy	XOrgHrDesc.xsd

Design Assumptions

- The REGIONALITY_HEAD table contains one record for each group/organizational hierarchy value that is defined for regionality. Regionality tables are used to define specific locations, suppliers, and/or departments that groups of users have responsibility for. These tables are not referenced within RMS, but can be used to customize reporting as well as on-line access if desired.
- Location trait records must exist prior to attaching them to any hierarchy.
- Chains do not have location traits associated with them.
- Some of the business validation is referential or involves uniqueness. This validation is handled automatically by the referential integrity constraints and the unique indexes implemented on the database.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
CHAIN	Yes	Yes	Yes	Yes
AREA	Yes	Yes	Yes	Yes
REGION	Yes	Yes	Yes	Yes
DISTRICT	Yes	Yes	Yes	Yes
LOC_AREA_TRAITS	Yes	Yes	No	Yes
LOC_REGION_TRAITS	Yes	Yes	No	Yes
LOC_DISTRICT_TRAITS	Yes	Yes	No	Yes
LOC_TRAITS_MATRIX	Yes	Yes	No	Yes
REGIONALITY_HEAD	Yes	No	No	No
REGIONALITY_DEPT	Yes	No	No	No
REGIONALITY_SUP_DEPT	Yes	No	No	No
REGIONALITY_SUP	Yes	No	No	No
REGIONALITY_TEMP	Yes	No	No	No

Payment Terms Subscription API

Functional Area

Payment Terms

Business Overview

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

Data Flow:

An external system will publish a payment term, thereby placing the payment term information onto the RIB. RMS will subscribe to the payment term information as published from the RIB and place the information onto RMS tables depending upon the validity of the records enclosed within the message.

Message Structure:

The payment term message will consist of a payment term record header and detail.

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

Package Impact

Filename: rmssub_ptrms/b.pls

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

All of the following procedures exist within RMSSUB_PAYTERM.

```
CONSUME
       (O_status_code              OUT          VARCHAR2,
        O_error_message            OUT          VARCHAR2,
        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 Terms messages. The valid message types for Terms messages are: paytermCre, paytermMod, paytermdtlCre and paytermdtlMod. 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_PAYTERM_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_PAYTERM_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.

Internal Procedure:

```
HANDLE_ERROR
            (O_status_code          IN OUT      VARCHAR2,
            O_error_message         IN OUT      RTK_ERRORS.RTK_TEXT%TYPE,
            I_cause                 IN          VARCHAR2,
            I_program               IN          VARCHAR2)
```

This is the standard error handling function that wraps the API_LIBRARY.HANDLE_ERROR function.

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

Business Validation Mode
Filename: rmssub_ptrmvals/b.pls

This function performs all business validation associated with Terms create and modify 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.

All of the following functions exist within RMSSUB_PAYTERM_VALIDATE.

```
CHECK_MESSAGE
           (O_error_message         OUT         RTK_ERRORS.RTK_TEXT%TYPE,
            O_dml_rec               OUT         TERMS_SQL.PAYTERM_REC,
            I_message               IN          "RIB_PayTermDesc_REC",
            I_message_type          IN          VARCHAR2)
```

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

This function calls CHECK_REQUIRED_FIELDS to make sure that all required fields are not NULL. CHECK_ENABLED is called to check for the validity of records with start_date_active and end_date_active with enabled flag. CHECK_TERMS_HEAD and CHECK_TERMS_DETAIL are called to check for header and detail records before inserting and updating TERMS_DEATIL table. Finally, the payment terms record used for DML is populated within the POPULATE_RECORD function and passed back to RMSSUB_PAYTERM.CONSUME.

Internal Functions:

CHECK_REQUIRED_FIELDS

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

POPULATE_RECORDS

This function populates the payment terms output record with the values sent in the message.

CHECK_ENABLED

This function in a loop checks for start_date_active and end_date_active with the enabled_flag setting from RIB_MESSAGE. Declare cursor to retrieve vdate from table period and another cursor to retrieve start_date_active and end_date_active for the terms and terms_seq inputted from TERMS_DETAIL table. In a loop assign terms_seq to a local variable. Open cursor to retrieve start_date_active and end_date_Active from

TERMS_DETAIL table. If terms_detail.start_date_active is after period.vdate and if enabled_flag from the rib message is 'Y', then raise program error. If end_date_active is < vdate and enabled_flag from the rib message is 'Y' then raise program error. If vdate > = start_date_active and <= end_date_active and enabled_flag is 'N' then raise a program error.

CHECK_TERMS_HEAD

This function will be responsible for checking TERMS_HEAD record before populating TERMS_DETAIL table for new terms record. Calling TERM_SQL.HEADER_EXISTS function will perform this check.

CHECK_TERMS_DETAIL

This function checks existence of terms_detail records before updating detail record. Calling TERM_SQL.DETAIL_EXISTS function will perform this check.

DML Module

Filename: rmssub_ptrm_sqls/b.pls

The following function exists within RMSSUB_PAYTERM_SQL.

```
PERSIST
          (O_error_message          OUT          RTK_ERRORS.RTK_TEXT%TYPE,
           I_message                IN           TERMS_SQL.PAYTERM_REC,
           I_message_type           IN           VARCHAR2)
```

Perform INSERT/UPDATE 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 header insert, populate the header record defined in the term_sql package and call the term_sql.insert_header function with this header record. For the message type indicating a header or a detail insert, call the term_sql.insert_detail function and pass to it the detail node from the message.

For the message type indicating a header update, populate the header record defined in the term_sql package and call the term_sql.update_header function with this header record. For the message type indicating a detail update, call the term_sql.update_detail function and pass to it the detail node from the message.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
PayTermCre	Payment Terms Create Message	PayTermDesc.xsd
PayTermMod	Payment Terms Modify Message	PayTermDesc.xsd
PayTermDtlCre	Payment Terms Detail Create Message	PayTermDesc.xsd
PayTermDtlMod	Payment Terms Detail Modify Message	PayTermDesc.xsd

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.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
TERMS_DETAIL	Yes	Yes	Yes	No
TERMS_HEAD	Yes	Yes	Yes	No

PO Subscription API

Functional Area

Purchase Orders

Business Overview

This subscription API is used to keep RMS in synchronization with an external system that is responsible for maintaining purchase orders.

It is assumed that externally generated non-EDI purchase orders are being interfaced expressly for the facilitation of inventory movement in RMS.

This API also default expenses and HTS, applies rounding, defaults inventory management parameters, applies bracket costs, updates OTB, and inserts a record into the deals queue.

This API allows external systems to create, edit, and delete purchase orders within RMS. These transactions are performed immediately on receiving the message receipt, so success or failure can be communicated to the calling application.

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

If the Purchase order is a Franchise PO (location is a Franchise store), a corresponding Franchise order is created along with the PO.

Package Impact

Filename: rmssub_xordersqls/b.pls

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

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

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

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

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

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

Filename: rmssub_xordervals/b.pls

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

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

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

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

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

ORDER Create

- Check required fields on both header and detail nodes.
- Verify order number does NOT already exist.
- Verify attributes in the message header are correct.
- Verify attributes in the message detail are correct.
- Verify that if the order is a Franchise PO, there should only be 1 franchise location for the order being created.

- Verify that if the order is other than a Franchise PO, no franchise location should exist in the message.
- Verify that if the order is a Franchise PO, the supplier must be a DSD Supplier.
- Verify that if the order is a Franchise PO, the items in the order should belong to a franchise location with the same costing location.
- Verify that item is within order range of the supplier.
- Verify that item/supplier and item/supp/country exist for a non-pack item.
- Verify that item/supplier and item/supp/country exist for all components of a pack item.
- Create item/supplier, item/supp/country and item/supp/country of manufacture if they don't exist for a pack item.
- Create item/supp/country/loc if it does not exist for an item/location.
- Create item/loc relation if not already exist, including creating item_loc_soh, item_supp_country_loc, and price_hist records. If a pack item is involved, these records will be created for all component items.
- Verify that the ordered quantity is within the Supplier's minimum and maximum limits if available.
- Populate record ORDER_REC with message data for both header and detail.

ORDER Modify

- Check required fields on the header node.
- Verify order number already exists.
- Verify attributes in the message header are correct.
- Verify attributes that cannot be modified are not changed.
- Verify that the ordered quantity is within the Supplier's minimum and maximum limits if available.
- If the order is a Franchise PO, the order cannot be reinstated.
- If the order is an Approved Franchise PO, it cannot be set back to worksheet.
- Update ORDLOC appropriately if closing or reinstating an order.
- Populate record ORDER_REC.ORDHEAD_ROW with message data.

ORDER DETAIL Create

- Check required fields on the detail node.
- Verify order number already exists.
- Verify order/item/loc does **not** already exist.
- Verify that if the order is a Franchise PO, there should only be 1 franchise location in the message and it should be the same as the existing order location.
- Verify that if the order is other than a Franchise PO, no franchise location should exist in the message.
- Verify that if the order is a Franchise PO, the items should have the same costing location.
- Verify that item/supplier and item/supp/country exist for a non-pack item.
- Verify that item/supplier and item/supp/country exist for all components of a pack item.
- Create item/supplier, item/supp/country and item/supp/country of Manufacture if they do not exist for a pack item.

- Create item/supp/country/loc if it does not exist for an item/location.
- Create item/loc relation if not already exists, including creating ITEM_LOC_SOH, ITEM_SUPP_COUNTRY_LOC, and PRICE_HIST records. If a pack item is involved, these records will be created for all component items.
- Verify that the ordered quantity is within the Supplier's minimum and maximum limits if available.
- Populate record ORDER_REC.ORDLOCS and optionally, ORDER_REC.ORDSKUS with message data.

ORDER DETAIL MODIFY

- Check required fields on the detail node.
- Verify order/item/loc already exists.
- Verify attributes that cannot be modified are not changed.
- If order quantity is reduced, verify the new order quantity is not below what has already been received plus what is being shipped or expected.
- If order quantity is modified, ensure that quantity pre-scaled is not updated if the order has been previously approved. Otherwise, quantity pre-scaled should always be equal to quantity ordered.
- If the order line is cancelled or reinstated via the indicators, calculate the new quantity buckets.
- Verify that the ordered quantity is within the Supplier's minimum and maximum limits if available.
- Populate record ORDER_REC.ORDLOCS and optionally, ORDER_REC.ORDSKUS with message data.

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

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

ORDER Delete

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

ORDER Detail Delete

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

Filename: rmssub_xordersqls/b.pls

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

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

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

ORDER Create

- Inserts records in the ORDHEAD, ORDSKU, ORDLOC tables.
- If the order is a Franchise PO, this inserts records in the WF_ORDER_HEAD and WF_ORDER_DETAIL tables.

ORDER Modify

- Updates a record in the ORDHEAD table.
- If the order is a Franchise PO, this updates records in the WF_ORDER_HEAD table.

ORDER Delete

- Delete an order from ORDHEAD, ORDSKU, ORDLOC tables.
- If the order is a Franchise PO, this deletes the corresponding record from the WF_ORDER_HEAD and WF_ORDER_DETAIL tables.

ORDER Detail Create

- Inserts records in the ORDLOC and optionally, ORDSKU tables.
- If the order is a Franchise PO, this inserts records in the WF_ORDER_HEAD and WF_ORDER_DETAIL tables.

ORDER Detail Modify

- Updates records in the ORDLOC and/or ORDSKU table.
- If the order is a Franchise PO, this updates records in the WF_ORDER_DETAIL table.
- Also verify it doesn't end up with an Approved order with 0 total order quantity.

ORDER Detail Delete

- Delete records from ORDLOC and optionally, ORDSKU tables.
- If the order is a Franchise PO, this deletes the corresponding record from the WF_ORDER_DETAIL table.
- Also verify it doesn't end up with an Approved order with no detail or with 0 total order quantity.

Call L10N Localization Decoupling Layer package (L10N_SQL) to determine if the order requires tax calculation. Same package will route call to 3rd party tax application if needed.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
XorderCre	Order Create Message	XOrderDesc.xsd
XorderMod	Order Modify Message	XOrderDesc.xsd
XorderDel	Order Delete Message	XOrderRef.xsd
XorderDtlCre	Order Detail Create Message	XOrderDesc.xsd
XorderDtlMod	Order Detail Modify Message	XOrderDesc.xsd
XorderDtlDel	Order Detail Delete Message	XOrderRef.xsd

Design Assumptions

Many ordering functionalities that are available on-line are not supported via this API. Deposit item functionality is not available in this API; that is to say a deposit contents item on the order does not automatically create the corresponding container item for the deposit item.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ORDHEAD	Yes	Yes	Yes	Yes
ORDSKU	Yes	Yes	Yes	Yes
ORDLOC	Yes	Yes	Yes	Yes
ITEM_SUPPLIER	Yes	Yes	No	No
ITEM_SUPP_COUNTRY	Yes	Yes	No	No
ITEM_SUPP_MANU_COUNTRY	Yes	Yes	No	No
ITEM_SUPP_COUNTRY_LOC	Yes	Yes	No	No
ITEM_LOC	Yes	Yes	No	No
ITEM_LOC_SOH	Yes	Yes	No	No
PRICE_HIST	No	Yes	No	No
ITEM_ZONE_PRICE	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
PACKITEM_BREAKOUT	Yes	No	No	No
SHIPMENT	Yes	No	No	No
SHIPSKU	Yes	No	No	No
APPT_DETAIL	Yes	No	No	No
ALLOC_HEADER	Yes	No	No	Yes
ALLOC_DETAIL	Yes	No	No	Yes

TABLE	SELECT	INSERT	UPDATE	DELETE
STORE	Yes	No	No	No
WAREHOUSE	Yes	No	No	No
SUPS	Yes	No	No	No
DEPS	Yes	No	No	No
CURRENCIES	Yes	No	No	No
CURRENCY_RATES	Yes	No	No	No
TERMS	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No
UNIT_OPTIONS	Yes	No	No	No
ADDR	Yes	No	No	No
WF_ORDER_HEAD	Yes	Yes	Yes	Yes
WF_ORDER_DETAIL	Yes	Yes	Yes	Yes

Receiving Subscription API

Functional Area

Receipt subscription:

- Purchase Order Receiving.
- Stock Order Receiving (including Transfers and Allocations).

Business Overview

RMS receives against purchase orders, transfers, and allocations. Transfers and allocations are collectively referred to as stock orders. The receipt subscription API processes carton-level receipts and a number of carton-level exceptions for stock orders receipts.

Purchase orders continue to be received only at the item level. If errors are encountered during purchase order receiving, the entire message is rejected and processing of the message stops.

Stock orders may be received at the bill of lading (BOL), carton, or item level. The following exceptions are automatically processed by the new stock order receiving package:

- Receiving against the wrong BOL.
- Receiving at a location which is a walk-through store for the intended location.
- Wrong store receiving.
- Unwanted cartons (those that have not been scanned).

Once RMS determines the appropriate receiving process for a carton, the shipment detail records are identified and existing line item level receiving is executed. The items are received into stock and transactions are updated.

Stock orders may be received at the BOL (receiving the entire shipment without checking the details), carton (receiving the entire carton on SHIPSKU without checking the details), or item level. When an error is encountered during stock order receiving, an error record is created for the BOL, carton, or item in error. Processing continues for the

remainder of the stock order receipt message. When the entire message has been processed, all of the error records are then handled. Error records are grouped together based on the type of error and a complete receipt message is created for each group. All errors will be collected in an error table, which will then be passed back to the RIB for further processing or hospitalization.

Carton-Level Receiving

The process for handling carton level receipts is as follows:

1. RMS determines whether a message type contains a receipt or an appointment.
2. If a receipt, RMS determines whether the document type is purchase order (P), transfer (T), or allocation (A).
3. If a stock order (transfer or allocation), RMS determines whether the receipt is an item level receipt (SK) or a carton level receipt (BL).
4. If a carton level receipt, two scenarios are possible. The message may contain (a) a bill of lading number but no carton numbers or (b) a bill of lading and one or more carton numbers.
   - Bill of lading/no cartons: RMS receives all cartons associated with the BOL along with their contents (line items).
   - Bill of lading/with cartons: RMS receives only the specified cartons and their contents (line items).
5. The status of the cartons determines how the cartons/items are processed. The status may be Actual (A), Overage (O), or Dummy BOL (D).

Actual (A)

The cartons are received at the correct location against the correct bill of lading.

Overage (O)

The carton does not belong to the current BOL. RMS attempts to match the contents with the correct BOL.

- If the carton belongs to a BOL at the given location, RMS receives the carton against the correct BOL at the given location.
- If the carton belongs to a BOL at a related walk-through store, RMS receives the carton against the intended BOL at the intended location.
- If the carton belongs to a BOL at an unrelated location, RMS uses the wrong store receiving process.

Dummy BOL (D)

Cartons were received under a dummy bill of lading (BOL) number. RMS attempts to match the contents with a valid BOL.

- If the carton belongs to a valid BOL at the given location, RMS receives the carton against the intended BOL at the given location.
- If the carton belongs to a valid BOL at a related walk-through store, RMS receives the carton against the intended BOL at the intended location.
- If the carton belongs to a valid BOL at an unrelated location, RMS uses the wrong store receiving process.

The wrong_st_receipt_ind system option controls whether wrong store receiving is available in RMS. The wrong_st_receipt_ind must be set to Y (Yes) to turn on this functionality. Wrong store receiving is done at the line item level. Inventory, average costs, and transactions for both the intended location and actual location are adjusted to accurately reflect the actual location of the items.

Doc Types

Receipts are processed based upon the document type indicator in the message. The indicator serves as a flag for RMSSUB_RECEIVE.CONSUME to use when calling the appropriate function that validates the data and writes the data to the base tables. The following are the document types and respective package and function names:

- A – for allocation. STOCK_ORDER_RCV_SQL.ALLOC_LINE_ITEM
- P – for purchase order. ORDER_RCV_SQL .PO_LINE_ITEM
- T – for transfer. STOCK_ORDER_RCV_SQL.TSF_LINE_ITEM

Blind Receipt Processing

A blind receipt is generated by an external application whenever a movement of goods is initiated by that application. RMS has no prior knowledge of blind receipts. RMS handles blind receipts when it runs STOCK_ORDER_RCV_SQL (transfers and allocations) or PO_RCV_SQL (purchase orders). If no appointment record exists on APPT_DETAIL, the respective function writes a record to the DOC_CLOSE_QUEUE table.

When a transfer, PO or allocation is received at a location, the external location (store or warehouse) will publish a receipt message to the RIB indicating that the stock has arrived. RMS will subscribe to the receipt message and update the appropriate tables, including shipment, transfer/allocation/purchase order, inventory and stock ledger.

For stock order receiving the ownership of the goods moves to the receiving location at the time of shipment. As a result, financial transaction records are written for the goods shipped when RMS processes a BOL message. At the receiving time, financial transaction records will only need to be written for the overage receiving. In addition, the stock order receiving process also handles the situations where stock is received with no receipt, or if the stock is received at wrong stores, or if the item received is on a dummy carton.

The receipt message is a hierarchical message that can contain a series of receipts. Each receipt corresponds to a transfer or an allocation or a PO, and can contain carton or item details. Purchase orders are only received at the item level. Any errors encountered during purchase order receiving will cause the entire message to be rejected and processing of the message will stop.

When receiving a customer order at stores, SIM will send a receipt message to both RMS and OMS, using a new message type of 'receiptordadd'. RMS will process 'receiptordadd' message in the same way as 'receiptadd'.

RMS supports of Brazil Localization. This includes a layer of code to enable decoupling of localization logic that is only required for country-specific configuration. This layer affects the RIB API flows including Receiving subscription.

L10N Localization Decoupling Layer:

This is a layer of code which enables decoupling of localization logic that is only required for certain country-specific configuration. This layer affects the RIB API flows including Receiving subscription. This allows RMS to be installed without requiring customers to install or use this localization functionality, where not required.

Package Impact

Filename: rmssub_receivings/b.pls

```
RMSSUB_RECEIVING.CONSUME
               (O_status_code         IN OUT         VARCHAR2,
                O_error_message       IN OUT         VARCHAR2,
                I_message             IN             RIB_OBJECT,
                I_message_type        IN             VARCHAR2,
                O_rib_otbdesc_rec     OUT            "RIB_OTBDesc_REC",
                O_rib_error_tbl       OUT            RIB_ERROR_TBL)
```

This procedure will make calls to receiving or appointment functions based on the value of I_message_type. If I_message type is RECEIPT_ADD or RECEIPT_UPD or RECEIPT_ORDADD, then a call is made to RMSSUB_RECEIPT.CONSUME, casting the message as a "RIB_ReceiptDesc_REC". If I_message_type is APPOINT_HDR_ADD, APPOINT_HDR_UPD, APPOINT_HDR_DEL, APPOINT_DTL_ADD, APPOINT_DTL_UPD, or APPOINT_DTL_DEL, then a call is made to RMSSUB_APPOINT.CONSUME. This is the procedure called by the RIB.

```
RMSSUB_RECEIVING.HANDLE_ERRORS
               (O_status_code         IN OUT         VARCHAR2,
                IO_error_message      IN OUT         VARCHAR2,
                I_cause               IN             VARCHAR2,
                I_program             IN             VARCHAR2)
```

Standard error handling function that wraps the API_LIBRARY.HANDLE_ERROR function.

Filename: rmssub_receipts/b.pls

```
RMSSUB_RECEIPT.CONSUME
               O_status_code          IN OUT         VARCHAR2,
               O_error_message        IN OUT         VARCHAR2,
               I_rib_receiptdesc_rec  IN             "RIB_ReceiptDesc_REC",
               I_message_type         IN             VARCHAR2,
               O_rib_otbdesc_rec      OUT            "RIB_OTBDesc_REC",
               O_rib_error_tbl        OUT            RIB_ERROR_TBL)
```

This function performs PO receiving and stock order receiving for each receipt in the message. Document type 'P' is for purchase order receiving, 'A' for allocation receiving, and 'T', 'V', 'D' for transfer receiving. All other document types are invalid.

The RIB object "RIB_ReceiptDesc_REC" is included in RIB_ReceiptOverage_REC" to accommodate for Overages.

Calls are made to ORDER_RCV_SQL.INIT_PO_ASN_LOC_GROUP, STOCK_ORDER_RCV_SQL.INIT_TSF_ALLOC_GROUP, and RMSSUB_RECEIPT_ERROR.INIT. These functions initialize global variables and clean out cached info.

- The process then loops through each receipt in the message and performs localization check. If localized, invoke localization logic through L10N_SQL decoupling layer for procedure key 'CONSUME_RECEIPT'. If not localized, call CONSUME_RECEIPT for normal processing:
- If the document type is 'P' (purchase order), it calls ORDER_RCV_SQL.PO_LINE_ITEM to receive the items on the PO.
- If the document type is 'T', 'D', 'V' (transfer) or 'A' (allocation), it calls RMSSUB_STKORD_RECEIPT.CONSUME to receive the items on the transfer or allocation.

- If the document type is not 'P', 'T', 'D', 'V' or 'A' the message processing is stopped and an error message returned.

After processing all receipts, call ORDER_RCV_SQL.FINISH_PO_ASN_LOC_GROUP, STOCK_ORDER_RCV_SQL.FINISH_TSF_ALLOC_GROUP, and RMSSUB_RECEIPT_ERROR.FINISH. These functions wrap up the processing for receiving and error logic.

If any records exist on the rib_otb_tbl returned by ORDER_RCV_SQL.FINISH_PO_ASN_LOC_GROUP, then create a rib_otbdesc_rec object and add the rib_otb_tbl to the object.

Filename: rmssub_stkord_receipts/b.pls

```
RMSSUB_STKORD_RECEIPT.CONSUME
              (O_status_code         IN OUT         VARCHAR2,
               O_error_message       IN OUT         RTK_ERRORS.RTK_TEXT%TYPE,
               I_appt                IN             APPT_HEAD.APPT%TYPE,
               I_rib_receipt_rec     IN             "RIB_Receipt_REC")
```

This function will process stock order receiving for all records within the rib_receipt_rec passed in. First, this function calls RMSSUB_RECEIPT_ERROR.BEGIN_RECEIPT. This function holds onto the header level information (appt_nbr and rib_receipt_rec), which may be used to create error objects.

Next, RMSSUB_RECEIPT_VALIDATE.CHECK_RECEIPT is called, which does validation at the receipt level. If the validation fails the receipt is rejected by calling RMSSUB_RECEIPT_ERROR.ADD_ERROR.

The package does carton-level receiving when receipt_type = 'BL', and item-level receiving when receipt_type = 'SK'.

There are two scenarios for carton-level receiving:

1. The rib_receipt_rec contains a bol_no and no cartons (no detail nodes). In this case the function RMSSUB_STKORD_RECEIPT_VALIDATE.CHECK_BOL is called, which does business level validation for the BOL. If the validation succeeds then RMSSUB_STKORD_RECEIPT_SQL.PERSIST_BOL is called. If the validation fails the BOL receipt is rejected by calling RMSSUB_RECEIPT_ERROR.ADD_ERROR.
2. The rib_receipt_rec contains a bol_no and 1 or more cartons (detail nodes). In this case, the process loops through each carton in the receipt and calls the function RMSSUB_STKORD_RECEIPT_VALIDATE.CHECK_CARTON. This function does business level validation for a carton. If the validation succeeds RMSSUB_STKORD_RECEIPT_SQL.PERSIST_CARTON is called. If the validation fails because the carton is a duplicate (by checking the returned validation_code), then the call to PERSIST_CARTON is skipped and processing continues. Duplicates are ignored with no error. If the validation fails for any other reason then the carton is rejected by calling RMSSUB_RECEIPT_ERROR.ADD_ERROR.

Item (SKU) Level Receiving:

If the receipt is item-level ('SK') the process loops through the detail records and calls the function RMSSUB_STKORD_RECEIPT_VALIDATE.CHECK_ITEM, which does business level validation for the item details. If the validation succeeds then RMSSUB_STKORD_RECEIPT_SQL.PERSIST_LINE_ITEM is called to execute existing line item receiving package calls. If the validation fails then the item is rejected by calling RMSSUB_RECEIPT_ERROR.ADD_ERROR.

When all details for the receipt have been processed, or if the entire receipt itself is rejected, then RMSSUB_RECEIPT_ERROR.END_RECEIPT is called. This function groups all similar errors and creates the appropriate error objects.

If a break to sell sellable item is on the message, a call to CHECK_ITEM and GET_ORDERABLE_ITEMS is made to convert the sellable to its orderable items. For a break to sell item, the orderable items are on the transfers, allocations, shipment, inventory and stock ledger.

Filename: rmssub_stkord_rct_vals/b.pls

```
RMSSUB_STKORD_RECEIPT_VALIDATE.CHECK_RECEIPT
               (O_error_message       IN OUT         RTK_ERRORS.RTK_TEXT%TYPE,
                O_valid               OUT            BOOLEAN,
                O_validation_code     OUT            VARCHAR2,
                I_rib_receipt_rec     IN             "RIB_Receipt_REC")
```

This function performs business validation for a receipt. If any of the validations fail then O_validation_error is populated with the specified error code and O_valid is set equal to FALSE. Otherwise, O_validation_error is left as NULL and O_valid is set equal to TRUE.

```
RMSSUB_STKORD_RECEIPT_VALIDATE.CHECK_BOL
           (O_error_message          IN OUT  RTK_ERRORS.RTK_TEXT%TYPE,
            O_valid                  IN OUT  BOOLEAN,
            O_validation_code        IN OUT  VARCHAR2,
            O_shipment               IN OUT  SHIPMENT.SHIPMENT%TYPE,
            O_item_table             IN OUT  STOCK_ORDER_RCV_SQL.ITEM_TAB,
            O_qty_expected_table     IN OUT  STOCK_ORDER_RCV_SQL.QTY_TAB,
            O_inv_status_table       IN OUT  STOCK_ORDER_RCV_SQL.INV_STATUS_TAB,
            O_carton_table           IN OUT  STOCK_ORDER_RCV_SQL.CARTON_TAB,
            O_distro_no_table        IN OUT  STOCK_ORDER_RCV_SQL.DISTRO_NO_TAB,
            O_tampered_ind_table     IN OUT  STOCK_ORDER_RCV_SQL.TAMPERED_IND_TAB,
    I_bol_no              IN  SHIPMENT.BOL_NO%TYPE,
    I_to_loc              IN  SHIPMENT.TO_LOC%TYPE)
```

This function performs business validation for receipts using BOL-level receiving. During validation this function selects data from the SHIPMENT and SHIPSKU tables and passes this information out through the parameters. This is done so that these tables do not have to be hit again during the receiving (persist) process. If any of the validations fail then O_validation_error is populated with the specified error code and O_valid is set equal to FALSE. Otherwise, O_validation_error is left as NULL and O_valid is set equal to TRUE.

```
RMSSUB_STKORD_RECEIPT_VALIDATE.CHECK_CARTON
       (O_error_message              IN OUT  RTK_ERRORS.RTK_TEXT%TYPE,
        O_valid                      IN OUT  BOOLEAN,
        O_validation_code            IN OUT  VARCHAR2,
        O_ctn_shipment               IN OUT  SHIPMENT.SHIPMENT%TYPE,
        O_ctn_to_loc                 IN OUT  SHIPMENT.TO_LOC%TYPE,
        O_ctn_bol_no                 IN OUT  SHIPMENT.BOL_NO%TYPE,
        O_item_table                 IN OUT  STOCK_ORDER_RCV_SQL.ITEM_TAB,
        O_qty_expected_table         IN OUT  STOCK_ORDER_RCV_SQL.QTY_TAB,
        O_inv_status_table           IN OUT  STOCK_ORDER_RCV_SQL.INV_STATUS_TAB,
        O_carton_table               IN OUT  STOCK_ORDER_RCV_SQL.CARTON_TAB,
        O_distro_no_table            IN OUT  STOCK_ORDER_RCV_SQL.DISTRO_NO_TAB,
        O_tampered_ind_table         IN OUT  STOCK_ORDER_RCV_SQL.TAMPERED_IND_TAB,
        O_wrong_store_ind            IN OUT  VARCHAR2,
        O_wrong_store                IN OUT  SHIPMENT.TO_LOC%TYPE,
        I_bol_no                     IN      SHIPMENT.BOL_NO%TYPE,
        I_to_loc                     IN      SHIPMENT.TO_LOC%TYPE,
        I_from_loc                   IN      SHIPMENT.FROM_LOC%TYPE,
        I_from_loc_type              IN      SHIPMENT.FROM_LOC_TYPE%TYPE,
        I_rib_receiptcartondtl_rec   IN      "RIB_ReceiptCartonDTL_REC")
```

This function performs business validation for receipts using carton-level receiving. Based on the carton status, a carton can be received to the intended store only, or as a dummy carton or to the walk-through store of the intended store.

During validation this function selects data from SHIPMENT and SHIPSKU tables and passes this information out through the parameters. This is done so that these tables do not have to be hit again during the receiving (persist) process. If any of the validations fail then O_validation_error is populated with the specified error code and O_valid is set equal to FALSE. Otherwise, O_validation_error is left as NULL and O_valid is set equal to TRUE.

```
RMSSUB_STKORD_RECEIPT_VALIDATE.CHECK_ITEM
               (O_error_message       IN OUT         RTK_ERRORS.RTK_TEXT%TYPE,
                O_valid               OUT            BOOLEAN,
                O_validation_code     OUT            VARCHAR2,
                I_distro_no           IN             SHIPSKU.DISTRO_NO%TYPE,
                I_dummy_carton_ind    IN             VARCHAR2)
```

This function performs business validation for item details. If any of the validations fail then O_validation_error is populated with the specified error code and O_valid is set equal to FALSE. Otherwise, O_validation_error is left as NULL and O_valid is set equal to TRUE.

```
RMSSUB_STKORD_RECEIPT_SQL.PERSIST_BOL
       (O_error_message              IN OUT         RTK_ERRORS.RTK_TEXT%TYPE,
        I_appt                       IN             APPT_HEAD.APPT%TYPE,
        I_doc_type                   IN             APPT_DETAIL.DOC_TYPE%TYPE,
        I_shipment                   IN             SHIPMENT.SHIPMENT%TYPE,
        I_to_loc                     IN             SHIPMENT.TO_LOC%TYPE,
        I_bol_no                     IN             SHIPMENT.BOL_NO%TYPE,
 I_item_table                IN             STOCK_ORDER_RCV_SQL.ITEM_TAB,
 I_qty_expected_table        IN             STOCK_ORDER_RCV_SQL.QTY_TAB,
 I_inv_status_table          IN             STOCK_ORDER_RCV_SQL.INV_STATUS_TAB,
 I_carton_table              IN             STOCK_ORDER_RCV_SQL.CARTON_TAB,
 I_distro_no_table           IN      STOCK_ORDER_RCV_SQL.DISTRO_NO_TAB,
 I_tampered_ind_table        IN             STOCK_ORDER_RCV_SQL.TAMPERED_IND_TAB)
```

This function calls STOCK_ORDER_RCV_SQL.TSF_BOL_CARTON (for transfers) and STOCK_ORDER_RCV_SQL.ALLOC_BOL_CARTON (for allocations) to perform BOL level receiving.

```
RMSSUB_STKORD_RECEIPT_SQL.PERSIST_CARTON
(O_error_message        IN OUT  RTK_ERRORS.RTK_TEXT%TYPE,
 I_appt                         IN      APPT_HEAD.APPT%TYPE,
 I_doc_type                     IN      APPT_DETAIL.DOC_TYPE%TYPE,
 I_shipment                     IN      SHIPMENT.SHIPMENT%TYPE,
 I_to_loc               IN      SHIPMENT.TO_LOC%TYPE,
 I_bol_no               IN      SHIPMENT.BOL_NO%TYPE,
 I_receipt_no                   IN      APPT_DETAIL.RECEIPT_NO%TYPE,
 I_disposition                  IN      INV_STATUS_CODES.INV_STATUS_CODE%TYPE,
 I_receipt_date         IN      SHIPMENT.RECEIVE_DATE%TYPE,
 I_item_table                   IN      STOCK_ORDER_RCV_SQL.ITEM_TAB,
 I_qty_expected_table           IN      STOCK_ORDER_RCV_SQL.QTY_TAB,
 I_weight               IN      ITEM_LOC_SOH.AVERAGE_WEIGHT%TYPE,
 I_weight_uom                   IN      UOM_CLASS.UOM%TYPE,
 I_inv_status_table             IN      STOCK_ORDER_RCV_SQL.INV_STATUS_TAB,
 I_carton_table         IN      STOCK_ORDER_RCV_SQL.CARTON_TAB,
 I_distro_no_table              IN      STOCK_ORDER_RCV_SQL.DISTRO_NO_TAB,
 I_tampered_ind_table           IN      STOCK_ORDER_RCV_SQL.TAMPERED_IND_TAB,
 I_wrong_store_ind              IN      VARCHAR2,
 I_wrong_store                  IN      SHIPMENT.TO_LOC%TYPE)
```

This function calls STOCK_ORDER_RCV_SQL.TSF_BOL_CARTON (for transfers) and STOCK_ORDER_RCV_SQL.ALLOC_BOL_CARTON (for allocations) to perform carton level receiving.

```
RMSSUB_STKORD_RECEIPT_SQL.PERSIST_LINE_ITEM
               (O_error_message      IN OUT         RTK_ERRORS.RTK_TEXT%TYPE,
                I_location           IN             SHIPMENT.TO_LOC%TYPE,
                I_bol_no             IN             SHIPMENT.BOL_NO%TYPE,
                I_distro_no          IN             SHIPSKU.DISTRO_NO%TYPE,
                I_distro_type        IN             VARCHAR2,
                I_appt               IN             APPT_HEAD.APPT%TYPE,
                I_rib_receiptdtl_rec IN             "RIB_ReceiptDTL_REC")
```

This function calls STOCK_ORDER_RCV_SQL.TSF_LINE_ITEM (for transfers) and STOCK_ORDER_RCV_SQL.ALLOC_LINE_ITEM (for allocations) to perform item level receiving.

Filename: rmssub_receipt_errors/b.pls

For each item or carton found to be in error during the receiving process, an error record is created. When all details for a receipt have been processed, the error records for that receipt are grouped by the error type. Error objects are collected in an error table, which is passed back to the RIB for additional processing. This type of error handling allows all valid records to be processed even when an invalid record is encountered.

```
RMSSUB_RECEIPT_ERROR.INIT
               (O_error_message      IN OUT  RTK_ERRORS.RTK_TEXT%TYPE)
```

This function initializes variables for error processing. It is called in the 'init' section of the RMSSUB_RECEIPT.CONSUME() function.

```
RMSSUB_RECEIPT_ERROR.BEGIN_RECEIPT
               (O_error_message      IN OUT         RTK_ERRORS.RTK_TEXT%TYPE,
                I_appt_nbr           IN             APPT_HEAD.APPT%TYPE,
                I_rib_receipt_rec    IN             "RIB_Receipt_REC")
```

This function is called once for each receipt within RMSSUB_STKORD_RECEIPT.CONSUME(). It copies the header information into the package level variables. This information is used when an error record is created.

```
RMSSUB_RECEIPT_ERROR.ADD_ERROR
               (O_error_message      IN OUT         RTK_ERRORS.RTK_TEXT%TYPE,
                I_error_type         IN             VARCHAR2,
                I_error_code         IN             VARCHAR2,
                I_error_desc         IN             VARCHAR2,
                I_error_level        IN             VARCHAR2,
                I_error_object       IN             RIB_OBJECT)
```

Used whenever an item or carton error occurs within the stock order receiving process. All calls to this function occur within RMSSUB_STKORD_RECEIPT.CONSUME.

Parameter explanation:

- O_error_message: any error message created if this function fails (EXCEPTION).
- I_error_type: either 'BL' for business logic process error, or 'SY' for system error. Currently, 'BL' type errors are limited to BOL/carton level business validation errors.
- I_error_code: a specific code to identify why/how the error occurred.
- I_error_desc: text description of the error.
- I_error_level: lets the package know how to cast the I_detail_rec. Valid values are 'RECEIPT', 'BOL', 'CARTON', 'ITEM'.

- I_detail_rec: record which is in error. May be a rib_receipt_rec (RECEIPT or BOL level), rib_receiptdtl_rec (ITEM level), or rib_receiptcartondtl_rec (CARTON level). This value will be cast based on I_error_level.

This function creates a new error record based on the error level passed in (casting the I_error_object appropriately). If the error level is RECEIPT or BOL, then a rib_receipt_rec is created. If the error level is CARTON, a rib_receiptcartondtl_rec is created. If error level is ITEM, a rib_receiptdtl_rec is created. After creating this error record, it is added to the table of error records.

```
RMSSUB_RECEIPT_ERROR.END_RECEIPT
              (O_error_message  IN OUT  RTK_ERRORS.RTK_TEXT%TYPE)
```

This function is called from RMSSUB_STKORD_RECEIPT.CONSUME when all details of a receipt have been processed. It takes all of the error records for this receipt and groups them according to the type of error. It then creates an error object for each error type, adding detail nodes for each error record. When this is finished, it adds all of the error records to the error table.

```
RMSSUB_RECEIPT_ERROR.FINISH
              (O_error_message       IN OUT         RTK_ERRORS.RTK_TEXT%TYPE,
               O_rib_error_tbl       OUT            RIB_ERROR_TBL)
```

If any errors exist on the package level error table then the error table is copied into the output parameter (O_rib_error_tbl), which in turn gets passed out to the RIB for further processing. This function is called in the 'finish' section of the RMSSUB_STKORD_RECEIPT.CONSUME function.

Filename: stkordrcvs/b.pls

```
STOCK_ORDER_RCV_SQL.TSF_BOL_CARTON
(O_error_message       IN OUT         RTK_ERRORS.RTK_TEXT%TYPE,
 I_appt                IN             APPT_HEAD.APPT%TYPE,
 I_shipment            IN             SHIPMENT.SHIPMENT%TYPE,
 I_to_loc              IN             SHIPMENT.TO_LOC%TYPE,
 I_bol_no              IN             SHIPMENT.BOL_NO%TYPE,
 I_receipt_no          IN             APPT_DETAIL.RECEIPT_NO%TYPE,
 I_disposition         IN             INV_STATUS_CODES.INV_STATUS_CODE%TYPE,
 I_tran_date           IN             PERIOD.VDATE%TYPE,
 I_item_table          IN             ITEM_TAB,
 I_qty_expected_table  IN             QTY_TAB,
 I_weight              IN             ITEM_LOC_SOH.AVERAGE_WEIGHT%TYPE,
 I_weight_uom          IN             UOM_CLASS.UOM%TYPE,
 I_inv_status_table    IN             INV_STATUS_TAB,
 I_carton_table        IN             CARTON_TAB,
 I_distro_no_table     IN             DISTRO_NO_TAB,
 I_tampered_ind_table  IN             TAMPERED_IND_TAB,
 I_wrong_store_ind     IN             VARCHAR2,
 I_wrong_store         IN             SHIPMENT.TO_LOC%TYPE)
```

This function performs the BOL or carton level receiving for a transfer. It does the following:

- Update shipment to received status along with the received date.
- For each item on the SHIPSKU, builds an API record for transferring the item. An orderable but non-sellable and non-inventory item cannot be transferred. The message contains physical locations, but a transfer created in RMS (non-'EG' type) contains virtual locations only. The physical locations are converted to virtual locations if necessary.
- Because an externally generated transfer (type 'EG') holds physical locations on TSFHEAD, and physical warehouses do **not** have transfer entities, this API does **not** support the receiving of an externally generated warehouse to warehouse transfer

when system option INTERCOMPANY_TSF_IND is 'Y'. However, it does allow store to warehouse 'EG' transfer, because it is assumed that store is sending merchandise to the virtual warehouse within the same channel, hence the same transfer entity.

- When receiving a transfer to a finisher location, all stock will be received into the available bucket regardless of the inventory disposition on the message.
- When system option WRONG_ST_RECEIPT is 'Y', stock can be received at a store not originally intended. Inventory and stock ledger is adjusted for both the intended and the actual receiving store.
- The received quantity on TSFDETAIL is updated. If it is a wrong store receiving, the reconciled quantity on TSFDETAIL is updated.
- The received quantity and received weight on SHIPSKU are updated. If SHIPSKU is not found, a new receipt is created.
- For an 'EG' type of transfer, the received quantity is distributed among the virtual locations of the physical location based on SHIPMENT_INV_FLOW, and the received quantity on SHIPMENT_INV_FLOW is updated.
- For an 'MRT' type of transfer, the received quantity on MRT_ITEM_LOC is updated.
- The table APPT_DETAIL is updated if an appointment exists for the transfer detail; otherwise, a record is inserted into DOC_CLOSE_QUEUE.
- A call to DETAIL_PROCESSING to perform the bulk of the transfer receiving logic, including moving inventory from the in transit to the stock on bucket for the receiving location is made. For overage receiving, the stock on hand is adjusted for both the sending and receiving locations, the av_cost for the receiving location is adjusted and records are written to the stock ledger.

```
        STOCK_ORDER_RCV_SQL.TSF_LINE_ITEM
        (O_error_message   IN OUT     RTK_ERRORS.RTK_TEXT%TYPE,
 I_loc                 IN      ITEM_LOC.LOC%TYPE,
 I_item                IN      ITEM_MASTER.ITEM%TYPE,
 I_qty                 IN      TRAN_DATA.UNITS%TYPE,
 I_weight              IN      ITEM_LOC_SOH.AVERAGE_WEIGHT%TYPE,
 I_weight_uom          IN      UOM_CLASS.UOM%TYPE,
 I_transaction_type    IN      VARCHAR2,
 I_tran_date           IN      PERIOD.VDATE%TYPE,
 I_receipt_number      IN      APPT_DETAIL.RECEIPT_NO%TYPE,
 I_bol_no              IN      SHIPMENT.BOL_NO%TYPE,
 I_appt                IN      APPT_HEAD.APPT%TYPE,
 I_carton              IN      SHIPSKU.CARTON%TYPE,
 I_distro_type         IN      VARCHAR2,
 I_distro_number       IN      TSFHEAD.TSF_NO%TYPE,
 I_disp                IN      INV_STATUS_CODES.INV_STATUS_CODE%TYPE,
 I_tampered_ind        IN      SHIPSKU.TAMPERED_IND%TYPE,
 I_dummy_carton_ind    IN      SYSTEM_OPTIONS.DUMMY_CARTON_IND%TYPE)
```

Similar to TSF_BOL_CARTON, this function performs transfer receiving for one line item. In addition, if the item is indicated as a dummy carton on the message, it writes staging records to the DUMMY_CARTON_STAGE table. The actual matching and receiving of dummy carton transfers is performed during the batch cycle via dummyctn.pc.

```
        STOCK_ORDER_RCV_SQL.ALLOC_BOL_CARTON
        (O_error_message              IN OUT        RTK_ERRORS.RTK_TEXT%TYPE,
         I_appt                       IN            APPT_HEAD.APPT%TYPE,
         I_shipment                   IN            SHIPMENT.SHIPMENT%TYPE,
         I_to_loc                     IN            SHIPMENT.TO_LOC%TYPE,
         I_bol_no                     IN            SHIPMENT.BOL_NO%TYPE,
         I_receipt_no                 IN            APPT_DETAIL.RECEIPT_NO%TYPE,
```

```
I_disposition                IN
      INV_STATUS_CODES.INV_STATUS_CODE%TYPE,
I_tran_date                  IN            PERIOD.VDATE%TYPE,
I_item_table                 IN            ITEM_TAB,
I_qty_expected_table         IN            QTY_TAB,
I_weight                     IN     ITEM_LOC_SOH.AVERAGE_WEIGHT%TYPE,
I_weight_uom                 IN            UOM_CLASS.UOM%TYPE,
I_inv_status_table           IN            INV_STATUS_TAB,
I_carton_table               IN            CARTON_TAB,
I_distro_no_table            IN            DISTRO_NO_TAB,
I_tampered_ind_table         IN            TAMPERED_IND_TAB,
I_wrong_store_ind            IN            VARCHAR2,
I_wrong_store                IN            SHIPMENT.TO_LOC%TYPE)
```

This function performs the BOL or carton level receiving for an allocation. It does the following:

- Updates the shipment to received status along with the received date.
- For each item on the SHIPSKU, builds an API record for allocating the item. An orderable but non-sellable and non-inventory item cannot be allocated.
- Validates that item is on the allocation.
- When system option WRONG_ST_RECEIPT is 'Y', stock can be received at a store not originally intended. Inventory and stock ledger are adjusted for both the intended and the actual receiving store.
- Validates that ALLOC_DETAIL exists. Updates received quantity on ALLOC_DETAIL. If it is a wrong store receiving, updates the reconciled quantity on ALLOC_DETAIL.
- Updates received quantity and received weight on SHIPSKU. If SHIPSKU is not found, creates a new receipt for that.
- Updates APPT_DETAIL if appointment exists for the allocation detail; otherwise, inserts into DOC_CLOSE_QUEUE.
- Calls DETAIL_PROCESSING to perform the bulk of the allocation receiving logic, including moving inventory from the in transit to the stock on bucket for the receiving location. For overage receiving, adjusts stock on hand for both the sending and receiving locations, adjusts av_cost for the receiving location and writes stock ledger.

```
STOCK_ORDER_RCV_SQL.ALLOC_LINE_ITEM
(O_error_message      IN OUT        RTK_ERRORS.RTK_TEXT%TYPE,
 I_loc                IN            ITEM_LOC.LOC%TYPE,
 I_item               IN            ITEM_MASTER.ITEM%TYPE,
 I_qty                IN            TRAN_DATA.UNITS%TYPE,
 I_weight             IN            ITEM_LOC_SOH.AVERAGE_WEIGHT%TYPE,
 I_weight_uom         IN            UOM_CLASS.UOM%TYPE,
 I_transaction_type   IN            VARCHAR2,
 I_tran_date          IN            PERIOD.VDATE%TYPE,
 I_receipt_number     IN            APPT_DETAIL.RECEIPT_NO%TYPE,
 I_bol_no             IN            SHIPMENT.BOL_NO%TYPE,
 I_appt               IN            APPT_HEAD.APPT%TYPE,
 I_carton             IN            SHIPSKU.CARTON%TYPE,
 I_distro_type        IN            VARCHAR2,
 I_distro_number      IN            ALLOC_HEADER.ALLOC_NO%TYPE,
 I_disp               IN            INV_STATUS_CODES.INV_STATUS_CODE%TYPE,
 I_tampered_ind       IN            SHIPSKU.TAMPERED_IND%TYPE,
 I_dummy_carton_ind   IN            SYSTEM_OPTIONS.DUMMY_CARTON_IND%TYPE)
```

Similar to ALLOC_BOL_CARTON, this function performs allocation receiving for one line item. In addition, if the item is indicated as a dummy carton on the message, it writes staging records to the DUMMY_CARTON_STAGE table. The actual matching and

receiving of dummy carton allocations is performed during the batch cycle via dummyctn.pc.

```
STOCK_ORDER_RCV_SQL.INIT_TSF_ALLOC_GROUP
              (O_error_message  IN OUT  RTK_ERRORS.RTK_TEXT%TYPE)
```

For performance reasons, bulk processing is used for stock order receiving. This function initializes global variables for bulk processing and populates system options.

```
STOCK_ORDER_RCV_SQL.FINISH_TSF_ALLOC_GROUP
              (O_error_message  IN OUT  RTK_ERRORS.RTK_TEXT%TYPE)
```

For performance reasons, bulk processing is used for stock order receiving. This function bulk updates APPT_DETAIL, bulk updates DOC_CLOSE_QUEUE and TRAN_DATA.

Filename: ordrcvs/b.pls

```
ORDER_RCV_SQL.PO_LINE_ITEM
       (O_error_message   IN OUT     rtk_errors.rtk_text%TYPE,
        I_loc             IN         item_loc.loc%TYPE,
        I_order_no        IN         ordhead.order_no%TYPE,
        I_item            IN         item_master.item%TYPE,
        I_qty             IN         tran_data.units%TYPE,
        I_tran_type       IN         VARCHAR2,
        I_tran_date       IN         DATE,
        I_receipt_number  IN         appt_detail.receipt_no%TYPE,
        I_asn             IN         shipment.asn%TYPE,
        I_appt            IN         appt_head.appt%TYPE,
        I_carton          IN         shipsku.carton%TYPE,
        I_distro_type     IN         VARCHAR2,
        I_distro_number   IN         alloc_header.alloc_no%TYPE,
        I_destination     IN         alloc_detail.to_loc%TYPE,
        I_disp            IN         inv_status_codes.inv_status_code%TYPE,
        I_unit_cost       IN         ordloc.unit_cost%TYPE,
        I_shipped_qty     IN         shipsku.qty_expected%TYPE,
        I_weight          IN         item_loc_soh.average_weight%TYPE,
        I_weight_uom      IN         UOM_CLASS.UOM%TYPE,
        I_online_ind      IN         VARCHAR2)
```

This function is called once for each PO line item received. It validates input and calls RCV_LINE_ITEM for each item/location.

- If the PO received is a cross-dock PO to a warehouse, an allocation must exist for the PO/allocation/item/warehouse combination. The message will contain a physical warehouse, whereas ALLOC_HEADER will contain a virtual warehouse.
- If the item is received to a physical warehouse, then this function calls the distribution logic to determine each item/virtual warehouse/quantity, and calls RCV_LINE_ITEM for each of these combinations.
- If a simple pack catch weight item is received, it also updates SHIPSKU weight received and weight received UOM.

```
ORDER_RCV_SQL.RCV_LINE_ITEM
(O_error_message      IN OUT        rtk_errors.rtk_text%TYPE,
 I_phy_loc            IN            item_loc.loc%TYPE,
 I_loc                IN            item_loc.loc%TYPE,
 I_loc_type           IN            item_loc.loc_type%TYPE,
 I_order_no           IN            ordhead.order_no%TYPE,
 I_item               IN            item_master.item%TYPE,
 I_qty                IN            tran_data.units%TYPE,
 I_tran_type          IN            VARCHAR2,
 I_tran_date          IN            DATE,
 I_receipt_number     IN            appt_detail.receipt_no%TYPE,
 I_asn                IN            shipment.asn%TYPE,
 I_appt               IN            appt_head.appt%TYPE,
 I_carton             IN            shipsku.carton%TYPE,
```

```
I_distro_type        IN            VARCHAR2,
I_distro_number      IN            tsfhead.tsf_no%TYPE,
I_destination        IN            alloc_detail.to_loc%TYPE,
I_disp               IN            inv_status_codes.inv_status_code%TYPE,
I_unit_cost          IN            ordloc.unit_cost%TYPE,
I_shipped_qty        IN            shipsku.qty_expected%TYPE,
I_weight             IN            item_loc_soh.average_weight%TYPE,
I_weight_uom         IN            UOM_CLASS.UOM%TYPE,
I_online_ind         IN            VARCHAR2)
```

This function is called for each item/location combination. It validates input and performs PO receiving logic for each item.

- Receiving (tran_type = 'R') must be against a valid approved order; adjustment (tran_type = 'A') must be against a valid approved or closed order.
- Item on the message may be a referential item. Get its transaction level item.
- An orderable, but non-sellable and non-inventory item cannot be received.
- For a deposit content item, its container item is also received and added to the order if not already on the order.
- Inserts or updates ORDLOC for quantity received.
- Updates APPT_DETAIL if appointment exists; otherwise, insert into DOC_CLOSE_QUEUE.
- Inserts or updates SHIPMENT to received status.
- Inserts or updates SHIPSKU for received quantity. If SHIPSKU.QTY_RECEIVED is updated, also updates INVC_MATCH_WKSHT.MATCH_TO_QTY.
- If no deals exist for this order/item/loc, then INVC_SQL.UPDATE_INVOICE is called to perform invoice matching logic.
- Updates average cost and stock on hand for the stock received. If a pack is on the order, the updates are performed for the component items.
- Writes TRAN_DATA records (tran code 20) for the stock received. If a pack is on the order, TRAN_DATA records are written for the component items.
- Writes SUP_DATA.
- Request tickets to be printed if location is a store.
- If this is an adjustment to a closed order, sets the status back to 'A'pproved.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
receiptcre	Receipt Create Message	ReceiptDesc.xsd
receiptordcre	Receipt Create Message	ReceiptDesc.xsd
receiptmod	Receipt Modify (Adjustment) Message	ReceiptDesc.xsd

Design Assumptions

1. The stock order subscription process supports the break-to-sell functionality. Transfers, allocations and shipments in RMS will only contain break to sell orderable items. Inventory adjustment and stock ledger will be performed on the orderable only, not the sellable.
2. The stock order and order subscription process supports the catch weight functionality. It is assumed that a break-to-sell sellable item cannot be a simple pack catch weight item.
3. An externally generated transfer will contain physical locations. When system options INTERCOMPANY_TSF_IND = 'Y', the stock order receiving process currently does **not** support the receiving of an externally generated transfer that involves a warehouse to warehouse transfer. This is because a physical location does **not** have transfer entities.
4. Wrong store receiving is not supported for franchise transactions.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
TSFHEAD	Yes	No	Yes	No
TSFDETAIL	Yes	Yes	Yes	No
ALLOC_HEADER	Yes	No	Yes	No
ALLOC_DETAIL	Yes	No	Yes	No
ORDHEAD	Yes	No	Yes	No
ORDSKU	Yes	Yes	Yes	No
ORDLOC	Yes	Yes	Yes	No
SHIPMENT	Yes	Yes	Yes	No
SHIPSKU	Yes	Yes	Yes	No
TRAN_DATA	No	Yes	No	No
SUP_DATA	No	Yes	No	No
ITEM_LOC_SOH	Yes	Yes	Yes	No
ITEM_LOC	Yes	Yes	No	No
ITEM_ZONE_PRICE	Yes	Yes	No	No
PRICE_HIST	No	Yes	No	No
SHIPITEM_INV_FLOW	Yes	Yes	Yes	No
MRT_ITEM_LOC	Yes	No	Yes	No
APPT_DETAIL	Yes	No	Yes	No
DOC_CLOSE_QUEUE	No	Yes	No	No
DUMMY_CARTON_STAGE	No	Yes	No	No
ALC_HEAD	Yes	Yes	Yes	No
CONTRACT_HEADER	Yes	No	Yes	No
CONTRACT_DETAIL	Yes	No	Yes	No

TABLE	SELECT	INSERT	UPDATE	DELETE
INVC_MATCH_WKSHT	Yes	No	Yes	No
INVC_HEAD	Yes	Yes	Yes	No
INVC_DETAIL	Yes	Yes	Yes	No
INVC_TOLERANCE	Yes	Yes	Yes	Yes
INVC_XREF	Yes	Yes	No	No
INVC_MATCH_VAT	Yes	Yes	Yes	No
TERMS	Yes	No	No	No
SUPS	Yes	No	No	No
VAT_REGION	Yes	No	No	No
DEPS	Yes	No	No	No
WEEK_DATA	Yes	No	No	No
MONTH_DATA	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
ITEM_SUPP_COUNTRY_LOC	Yes	Yes	No	No
ITEM_SUPP_COUNTRY_DIM	Yes	No	No	No
UOM_CLASS	Yes	No	No	No
NWP	Yes	Yes	Yes	No
STORE	Yes	No	No	No
WH	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
ITEM_XFORM_HEAD	Yes	No	No	No
ITEM_XFORM_DETAIL	Yes	No	No	No
CURRENCIES	Yes	No	No	No
CURRENCY_RATES	Yes	No	No	No
PERIOD	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No

RTV Subscription API

Functional Area

Return to Vendor

Business Overview

RMS subscribes to return-to-vendor (RTV) messages from the RIB. When an RTV is shipped out from a warehouse or store, the RTV information is sent from the external system (such as RWMS and SIM) to the RIB. RMS subscribes to the RTV information as published from the RIB and places the information onto RMS tables, depending on the validity of the records enclosed within the message.

The RTV message can be processed as a flat message when the header description contains information for one RTV item. The message can also be processed as a hierarchical message when the detail node is populated with one or more RTV items. RMS primarily uses these messages to update inventory quantities and stock ledger values.

L10N Localization Decoupling Layer:

This is a layer of code which enables decoupling of localization logic that is only required for certain country-specific configuration. This layer affects the RIB API flows including RTV subscription. This allows RMS to be installed without requiring customers to install or use this localization functionality, where not required.

Package Impact

Filename: rmssub_rtvs/b.pls

```
RMSSUB_RTV.CONSUME
               (O_status_code         IN OUT         VARCHAR2,
                O_error_message       IN OUT         VARCHAR2,
                I_message             IN             RIB_OBJECT,
                I_message_type        IN             VARCHAR2)
```

This procedure initially ensures that the passed in message type is a valid type for RTV messages. The valid message types for RTV messages are listed in the Message XSD section below.

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

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

If the downcast is successful, then consume parses the message, verifies that the message passes all of RMS's business validation and persists the information to the RMS database. It does this by calling CONSUME_RTV.

```
RMSSUB_RTV.CONSUME_RTV
               O_status_code       IN OUT  VARCHAR2,
               O_error_message     IN OUT  VARCHAR2,
               I_message           IN      RIB_OBJECT,
               I_message_type      IN      VARCHAR2,
               I_check_l10n_ind    IN      VARCHAR2)
            Performs localization check. If localized, invoke RFM's logic
            through L10N_SQL decoupling layer for procedure key 'CONSUME_RTV'. If
            not localized, call CONSUME_RTV for normal processing.
RMSSUB_RTV.CONSUME_RTV
               (O_error_message     IN OUT  VARCHAR2,
               IO_L10N_RIB_REC     IN OUT  L10N_OBJ)
               Public function to call RMSSUB_RTV.CONSUME_RTV_CORE.
               RMSSUB_RTV.CONSUME_RTV_CORE
               (O_error_message     IN OUT  VARCHAR2,
               I_message           IN      RIB_OBJECT,
               I_message_type      IN      VARCHAR2)
```

This function contains the main processing logic:

If the downcast is successful, then consume calls PARSE_RTV to parse the RTV message and PROCESS_RTV to perform business validation and desired functionality. Any time the message fails business validation, a status of "E" is returned to the external system along with an appropriate error message.

Once the message has been successfully processed, a success status, "S", is returned to the external system indicating that the message has been successfully received and persisted to the RMS database.

PARSE_RTV

This function parses the RIB_OBJECT and builds an API rtv_record for processing.

Gross cost can be included in the detail RIB_RTVDtl_REC. If the gross cost is present, then it is stored as unit_cost and unit_cost is stored as extended_base_cost.

Jurisdiction code also is determined based on supplier.

PROCESS_RTV

This function calls RTV_SQL.APPLY_PROCESS to perform all business validation and desired functionality associated with a RTV message.

For break to sell items, if a sellable only item is on the message, CHECK_ITEMS and GET_ORDERABLE_ITEMS are called to convert the sellable item(s) to the corresponding orderable item(s). The orderable items are inserted or updated on the tables affected by an RTV.

The RTV_SQL.APPLY_PROCESS is called for each of the orderable items and each of the regular items.

CHECK_ITEMS

This function separates the item details on the message into two groups: one contains sellable only items and one contains regular items.

GET_ORDERABLE_ITEMS

This function builds a collection of orderable items based on the sellable items. It calls ITEM_XFORM_SQL.RTV_ORDERABLE_ITEM_INFO to distribute the sellable quantities among the orderable items.

Filename: rtvs/b.pls

RTV_SQL.APPLY_PROCESS

This function performs business validation and desired functionality for a RTV message. It includes the following:

- Verifies that an orderable but non-sellable and non-inventory item **cannot** be an RTV item.
- Verifies that an RTV item must be a tran-level or above tran-level item.
- If the RTV item is a simple pack catch weight item, verifies that weight and weight unit of measure (UOM) are either both defined or both NULL, and weight UOM is in the MASS UOM class.
- Verifies that the item supplier relation exists.
- Verifies that the location is a valid store or warehouse.
- Verifies that the item/loc relation exists.
- If returning a pack to a warehouse, the pack must be received as pack at the warehouse.
- Verifies that from disposition is a valid inventory status code (on INV_STATUS_CODES).
- Verifies that the reason code is a valid RTV reason code (code type 'RTVR' on CODE_DETAIL).
- For an externally generated RTV, if the location is a warehouse, then physical location is on the message. RTV quantity will be distributed among the virtual locations of the physical location.
- Checks for the existence of RTV in RTV_HEAD based on: a) rtv_order_no; b) ext_ref_no and location. An RTV is updated if it already exists and inserted if not. The RTV is marked as shipped.
- Checks for the existence of RTV item in RTV_DETAIL based on: rtv_order_no, item, reason and inventory status. An RTV_DETAIL is updated if it already exists and inserted if not.
- If the RTV item is a content item of a deposit item, RTV_DETAIL is inserted or updated for the associated container item.
- Determines RTV unit cost as the following:
  - Uses the unit cost on the RTV message if defined. It is in location currency. Otherwise.
  - Uses RTV_DETAIL.unit_cost if exists. It is in supplier currency. Otherwise.
  - Uses the last receipt cost if exists. It is in location currency. Otherwise.
  - Uses item's WAC at the location. It is in location currency.
  - The unit cost is used to evaluate the cost of the RTV goods. The cost values on RTV tables are written in supplier currency, but all TRAN_DATA records are written in location currency.
- If the RTV item is a simple pack catch weight item, the total RTV cost is based on weight.
- Updates the following stock buckets on ITEM_LOC_SOH: RTV_QTY, STOCK_ON_HAND, PACK_COMP_SOH. For a simple pack catch weight item at the warehouse, also updates average weight.
- Writes the following TRAN_DATA records:
  - **24** – for RTV. It writes units, total_cost and total_retail.

- **71/72** – for cost variance between item's WAC at the location and RTV unit cost. It writes units and total_cost.
- **65** – for restocking fees. For a non-MRT type of RTV, the restocking fee is written for the RTV location. For an MRT type of RTV, the restocking fee is distributed among the MRT locations. It writes units and total_cost.
- **22** – for stock adjustment, if stock counting has already happened at the store for the item.

  If the RTV item is a pack, TRAN_DATA is written for component items. If the RTV location is a physical warehouse, TRAN_DATA is written for virtual locations. TRAN_DATA total cost and total retail are always written in location currency.

- Creates or updates INVC_HEAD and INVC_DETAIL for the RTV.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
rtvcre	RTV Create Message	RTVDesc.xsd

Design Assumptions

1. Catch weight functionality is **not** applied to the following areas:
   - Any of the retail calculations (including total_retail on TRAN_DATA and retail markup/markdown).
   - The total amount on SUP_DATA.
   - Open to buy buckets.
   - When a catch weight component item's standard UOM is a MASS UOM, TRAN_DATA.units is based on V_PACKSKU_QTY.qty instead of the actual weight.
2. MRT RTV can only be created in RMS. Therefore it will only contain virtual locations. Physical location distribution logic does **not** apply to MRT RTVs.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
RTV_HEAD	Yes	Yes	Yes	No
RTV_DETAIL	Yes	Yes	Yes	No
ITEM_LOC_SOH	Yes	No	Yes	No
TRAN_DATA	No	Yes	No	No
INV_STATUS_CODES	Yes	No	No	No
CODE_DETAIL	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
ITEM_SUPPLIER	Yes	No	No	No

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_SUPP_COUNTRY	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
STORE	Yes	No	No	No
WH	Yes	No	No	No
SHIPMENT	Yes	No	No	No
SHIPSKU	Yes	No	No	No
DEPS	Yes	No	No	No
SUPS	Yes	No	No	No
ADDR	Yes	No	No	No
UOM_CLASS	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
MRT_ITEM_LOC	Yes	No	No	No
ITEM_XFORM_HEAD	Yes	No	No	No
ITEM_XFORM_DETAIL	Yes	No	No	No
INVC_HEAD	Yes	Yes	Yes	Yes
INVC_DETAIL	Yes	Yes	No	Yes
INVC_NON_MERCH	No	Yes	No	Yes
INVC_MERCH_VAT	Yes	Yes	Yes	Yes
INVC_DETAIL_VAT	Yes	No	No	Yes
INVC_MATCH_QUEUE	Yes	No	No	Yes
INVC_DISCOUNT	Yes	No	No	Yes
INVC_TOLERANCE	Yes	No	No	Yes
ORDLOC_INVC_COST	Yes	No	Yes	No
NON_MERCH_CODE_HEAD	Yes	No	No	No

Stock Order Status Subscription API

Functional Area

Stock Order Status

Business Overview

A stock order is an outbound merchandise request from a warehouse or store. In RMS, a stock order takes the form of either a transfer or allocation. RMS subscribes to stock order status messages from the RIB, published by an external application, such as a store system (SIM, for example) or a warehouse management system (RWMS, for example) to communicate the status of a specific stock order. This communication provides for the synchronization of data between RWMS/SIM and RMS. The information from RWMS and SIM has only one level, in other words no detail records. RMS uses the data contained in the messages to:

- Update the following tables when the status of the 'distro' changes at the store or warehouse:
  - ALLOC_DETAIL
  - ITEM_LOC_SOH
  - TSFDETAIL
- To determine when the store or warehouse is processing a transfer or allocation. In-process transfers or allocations cannot be edited and are determined by the initial and final quantities to be filled by the external system.
- When RMS is integrated with an external Order Management System (OMS), OMS will subscribe to SOStatus messages published from SIM and WMS when a store or warehouse cannot fulfill a customer order. OMS, in turn, sends a customer order cancellation request to RMS. In order to prevent duplicate processing for the same cancellation message, this subscription API will ignore 'no inventory' statuses received from RWMS and SIM for a customer order transfer.

Stock Order Status Explanations

The following tables describe the stock order statuses for both transfers and allocation document types and what occurs in RMS after receiving the respective status. Document_types of 'T', 'D' and 'S' indicate if the transfer is initiated in RMS, a warehouse system, or a store system respectively. Statuses other than listed below are ignored by RMS.

Stock order status received in message on a transfer where 'distro_document_type' = 'T', 'D', 'S')	What RMS does
SI (Stock Increased) When SIM or RWMS publishes a message on a transfer with a status of SI (Stock Increased), RMS will insert or update TSFDETAIL for the transfer/item combination.	Insert or increase tsfdetail.tsf_qty Increase item_loc_soh.tsf_reserved_qty for the from location and item_loc_soh.tsf_expected_qty for the to location

Stock order status received in message on a transfer where 'distro_document_type' = 'T', 'D', 'S')	What RMS does
SD (Stock Decreased) When SIM or RWMS publishes a message on a transfer with a status of SD (Stock Decreased), RMS will delete or update TSFDETAIL for the transfer/item combination.	Delete or decrease tsfdetail.tsf_qty. Decrease item_loc_soh.tsf_reserved_qty for the from location and item_loc_soh.tsf_expected_qty for the to location
DS (Details Selected) When RWMS publishes a message on a transfer with a status of DS (Details Selected), RMS will increase the selected quantity on TSFDETAIL for the transfer/item combination.	Increase tsfdetail.selected_qty
DU (Details Un-selected) When RWMS publishes a message on a transfer with a status of DU (Details Un-Selected), RMS decreases the selected quantity on TSFDETAIL for the transfer/item combination.	Decrease tsfdetail.selected_qty
NI (WMS Line Cancellation) When RWMS publishes a message on a transfer with a status of NI (No Inventory – WMS Line Cancellation), RMS will decrease the selected quantity by the quantity on the message. RMS will also increase the cancelled quantity, decrease the transfer quantity, decrease the reserved quantity* for the from location, and decrease the expected quantity* for the to location by the lesser of 1). the quantity on the message; 2). the transfer quantity – shipped quantity. *If the transfer status is not Closed.	Decrease tsfdetail.selected_qty and tsfdetail.tsf_qty, increase tsfdetail.cancelled_qty, decrease item_loc_soh.tsf_reserved_qty for the from location and item_loc_soh.tsf_expected_qty for the from location Put transfer on doc_close_queue
PP (Distributed) When RWMS publishes a message on a transfer with a status of PP (Pending Pick - Distributed), RMS will decrease the selected quantity and increase the distro quantity.	Decrease tsfdetail.selected_qty, increase tsfdetail.distro_qty
PU (Un-Distribute) When RWMS publishes a message on a transfer with a status of PU (Un-Distribute), RMS will decrease the distributed qty.	Decrease tsfdetail.distro_qty
RS (Return To Stock) When RWMS published a message on a transfer with a status of RS (Return To Stock), RMS will decrease the distributed qty. RMS will also increase the cancelled quantity, decrease the transfer quantity, decrease the reserved quantity* for the from location, and decrease the expected quantity* for the to location by the lesser of 1). the quantity on the message; 2). the transfer quantity – shipped quantity. *If the transfer status is not Closed.*	Decrease tsfdetail.distro_qty and tsfdetail.tsf_qty, increase tsfdetail.cancelled_qty, decrease item_loc_soh.tsf_reserved_qty for the from location and item_loc_soh.tsf_expected_qty for the from location

Stock order status received in message on a transfer where 'distro_document_type' = 'T', 'D', 'S')	What RMS does
EX (Expired) When RWMS publishes a message on a transfer with a status of EX (Expired), RMS will increase the cancelled quantity, decrease the transfer quantity, decrease the reserved quantity* for the from location, and decrease the expected quantity* for the to location by the lesser of 1). the quantity on the message; 2). the transfer quantity – shipped quantity. **If the transfer status is not Closed.*	Increase tsfdetail.cancelled_qty, decrease tsfdetail.tsf_qty, item_loc_soh.tsf_reserved_qty for the from location and item_loc_soh.tsf_expected_qty for the To location Put transfer on doc_close_queue
SR (Store Reassign) When RWMS publishes a message on a transfer with a status of SR (Store Reassign) the quantity can be either positive or negative. In either case it will be added to the distro_qty (adding a negative will have the same effect as subtracting it).	Add to tsfdetail.distro_qty

Stock order status received in message on an ALLOCATION (where 'distro_document_type' = 'A')	What RMS does
SI (Stock Increased) When SIM or RWMS publishes a message on an allocation with a status of SI (Stock Increased), RMS will increase ALLOC_DETAIL for the allocation/item combination.	Increase alloc_detail.qty_allocated Increase item_loc_soh.tsf_reserved_qty for the from location and item_loc_soh.tsf_expected_qty for the To location
SD (Stock Decreased) When SIM or RWMS publishes a message on an allocation with a status of SD (Stock Decreased), RMS will decrease ALLOC_DETAIL for the allocation/item combination.	Decrease alloc_detail.qty_allocated. Decrease item_loc_soh.tsf_reserved_qty for the from location and item_loc_soh.tsf_expected_qty for the To location
DS (Details Selected) When RWMS publishes a message on an allocation with a status of DS (Details Selected), RMS will increase the selected quantity on alloc_detail for the allocation/item/location combination.	Increase alloc_detail.selected_qty
DU (Details Un-Selected) When RWMS publishes a message on an allocation with a status of DU (Details Un-Selected), RMS will decrease the selected quantity on alloc_detail for the allocation/item combination.	Decrease alloc_detail.selected_qty

Stock order status received in message on an ALLOCATION (where 'distro_document_type' = 'A')	What RMS does
NI (WMS Line Cancellation) When RWMS publishes a message on an allocation with a status of NI (No Inventory – WMS Line Cancellation), RMS will decrease the selected quantity by the quantity on the message. RMS will also increase the cancelled quantity, decrease the allocated quantity, decrease the reserved quantity* for the from location, and decrease the expected quantity* for the to location by the lesser of 1). the quantity on the message; 2). the allocation quantity – shipped quantity. **If the allocation status is not Closed and the allocation is a stand alone allocation.*	Decrease alloc_detail.qty_ selected and alloc_detail.qty_allocated, increase alloc_detail.cancelled_qty, decrease item_loc_soh.tsf_reserved_qty for the from location and item_loc_soh.tsf_expected_qty for the to location Put allocation on doc_close_queue
PP (Distributed) When RWMS publishes a message on an allocation with a status of PP (Pending Pick - Distributed), RMS will decrement the selected quantity and increment the distro quantity.	Decrease alloc_detail.qty_selecteded, increase alloc_detail.qty_distro
PU (Un-Distribute) When RWMS publishes a message on an allocation with a status of PU (Un-Distribute), RMS will decrease the distributed qty.	Decrease alloc_detail.qty_distro
RS (Return to Stock) When RWMS published a message on an allocation with a status of RS (Return to Stock), RMS will decrease the distributed qty. RMS will also increase the cancelled quantity, decrease the allocated quantity, decrease the reserved quantity* for the from location, and decrease the expected quantity* for the to location by the lesser of 1). the quantity on the message; 2). the allocation quantity – shipped quantity. **If the allocation status is not Closed and the allocation is a stand alone allocation.*	Decrease alloc_detail.qty_distro and alloc_detail.qty_allocated, increase alloc_detail.cancelled_qty, decrease item_loc_soh.tsf_reserved_qty for the from location and item_loc_soh.tsf_expected_qty for the to location
EX (Expired) When RWMS publishes a message on an allocation with a status of EX (Expired), RMS will increase the cancelled quantity, decrease the allocated quantity, decrease the reserved quantity* for the from location, and decrease the expected quantity* for the to location by the lesser of 1). the quantity on the message; 2). the transfer quantity – shipped quantity. **If the allocation status is not Closed and the allocation is a stand alone allocation.*	Decrease alloc_detail.qty_allocated, increase alloc_detail.qty_cancelled, decrease item_loc_soh.tsf_reserved_qty for the from location and item_loc_soh.tsf_expected_qty for the to location Put allocation on doc_close_queue
SR (Store Reassign) When RWMS publishes a message on an allocation with a status of SR (Store Reassign) the quantity can be either positive or negative. In either case, it will be added to the qty_distro (adding a negative will have the same affect as subtracting it).	Add to alloc_detail.qty_distro

Pack Considerations

Whenever the from location is a warehouse, a check if the item is a pack or an each is performed. If the item is not a pack item, no special considerations are necessary. For each warehouse-pack item combination, the receive_as_type on ITEM_LOC is checked to determine if it is received into the warehouse as a pack or a component item. If it is received as an each, ITEM_LOC_SOH for the component item is updated. If it is received as a pack, ITEM_LOC_SOH for the pack item and the component item are updated.

Package Impact

Filename: rmssub_sostatuss/b.pls

CONSUME

```
RMSSUB_SOSTATUS.CONSUME
               (O_status_code         IN OUT         VARCHAR2,
                O_error_message       IN OUT         VARCHAR2,
                I_message             IN             "RIB_SOStatusDesc_REC",
                I_message_type        IN             VARCHAR2);
```

This procedure accepts Stock Order Status information in the form of an Oracle Object data type from the RIB (I_message) and a message type of 'sostatuscre'. The procedure first calls the RESET function to initialize internal variables. The procedure then extracts the values from the oracle object. These are then passed on to private internal functions which validate the values and place them on the database depending upon the success of the validation.

BUILD_XTSFDESC

This function builds a RIB_XTsfDesc_REC object to be passed in the RMSSUB_XTSF.CONSUME function.

HANDLE_ERRORS

```
HANDLE_ERRORS(O_status                IN OUT    VARCHAR2,
              IO_error_message        IN OUT    RTK_ERRORS.RTK_TEXT%TYPE,
              I_cause         IN        VARCHAR2,
              I_program           IN    VARCHAR2);
```

If an error occurs in this procedure or any of the internal functions, this procedure places a call to HANDLE_ERRORS in order to parse a complete error message and pass back a status to the RIB.

This function is used to put error handling in one place in order to make future error handling enhancements easier to implement. The function consists of a call to API_LIBRARY.HANDLE_ERRORS. API_LIBRARY.HANDLE_ERRORS accepts a program name, the cause of the error and potentially an unparsed error message if one has been created through a call to SQL_LIB.CREATE_MESSAGE. The function uses these input variables to parse a complete error message and pass back a status, depending upon the message and error type, back up through the consume function and up to the RIB.

PARSE_SOS

This function first calls VALIDATE to check that the transfer or allocation from the oracle object exists in RMS. If the transfer or allocation exists, the function breaks down the message into its component parts and sends these parts into PROCESS_SOS. For customer order transfers, the customer order number and fulfill order number is also validated against the corresponding record in ORDCUST.

When RMS is integrated to OMS, this function skips processing for 'NI', 'EX', 'SI', 'SD', 'PP', 'PU' statuses received from RWMS and SIM for customer order transfers.

PROCESS_SOS

Based on the status sent from RWMS and SIM, quantity fields on either TSFDETAIL or ALLOC_DETAIL and ITEM_LOC_SOH are updated.

VALIDATE

Validates the distro is valid. A distro refers to either a transfer or an allocation.

UPDATE_TSF

Updates the record on TSFDETAIL, if the message is for a transfer.

UPDATE_ALLOC

Updates the record on ALLOC_DETAIL, if the message is for an allocation.

UPD_FROM_ITEM_LOC

Updates item_loc_soh.tsf_reserved_qty for the From Location. If the comp_level_upd indicator is 'Y' then it will also update the item_loc_soh.pack_comp_resv field for the item passed in.

UPD_TO_ITEM_LOC

Updates item_loc_soh.tsf_expected_qty for the To Location. If the comp_level_upd indicator is 'Y' then it will also update the item_loc_soh.pack_comp_exp field for the item passed in.

GET_RECEIVE_AS_TYPE

This function gets the Receive as type value from ITEM_LOC for the passed-in item and location combination.

POPULATE_DOC_CLOSE_QUEUE

This function is called to populate an array which holds stock order information that will be placed on the DOC_CLOSE_QUEUE table.

RESET

This function deletes any values that are currently held in the package's global variables.

DO_BULK

This function is used to do bulk inserts or updates of the ALLOC_DETAIL, TSFDETAIL, TSFHEAD and DOC_CLOSE_QUEUE tables. The tables are updated/inserted using the arrays that were built in the rest of the package.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
sostatuscre	Stock Order Status Create Message	SOStatusDesc.xsd

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.
- SOStatus supports transfers and allocations linked to a franchise order or return. For an existing transfer and allocation modified by a stock order status message, the quantity change is NOT reflected on the franchise order or return since the franchise order or return would have been approved already.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_LOC_SOH	Yes	Yes	Yes	No
ITEM_LOC	Yes	No	No	No
ALLOC_DETAIL	Yes	No	Yes	No
ALLOC_HEADER	Yes	No	No	No
TSFDETAIL	Yes	No	Yes	No
TSFHEAD	Yes	No	Yes	No
DOC_CLOSE_QUEUE	No	Yes	No	No
ORDCUST	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
WF_ORDER_HEAD	Yes	Yes	No	No
WF_ORDER_DETAIL	No	Yes	No	No
WF_ORDER_EXP	No	Yes	No	No

Stock Count Schedule Subscription API

Functional Area

Inventory – Stock Counts

Business Overview

Stock count schedule messages are published to the RIB by an integration subsystem, such as a store inventory management system, to communicate unit and value stock count schedules to RMS. RMS uses stock count schedule data to help synchronize the inventories of another application and RMS. The other application performs a physical inventory count and uploads the results, and RMS compares the discrepancies.

This API allows external systems to create, update, and delete stock counts within RMS. Only Unit and Value stock counts (stocktake_type = 'B') are subscribed by RMS at this time. Department, class and subclass can be null; if not provided a full count is presumed.

If the other application requires at year-end to consolidate annual and booking numbers, the annual count can be initiated by the other application and uploaded into RMS. RMS accepts the unit variances and processes these automatically. The financial values will need user input from the central office.

Package Impact

Filename: rmssub_stakeschedules/b.pls

```
CONSUME(O_status_code     IN OUT     VARCHAR2,
        O_error_message   IN OUT     VARCHAR2,
        I_message         IN         RIB_OBJECT,
        I_message_type    IN         VARCHAR2);
```

This package is used to subscribe to stock count schedule message, parse the details, and pass them into the stock schedule package.

- If the message type is StkCountSchDel, validates before deleting the cycle count.
- For other message types, business validations are performed before creating or updating the cycle count.
- Once the message has been successfully processed, 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.

Filename: stake_schedules/b.pls

This package is used to validate stock schedule data and insert/update to the stock count tables.

VALIDATE_VALUES

- Cannot delete a cycle count if it has been processed.
- Cannot update a cycle count that has started or has been set to be deleted.
- Cannot process anything if stock count is currently locked.

VALIDATE_HIERARCHY

- Unit and Value stock counts at a warehouse must be at the department level only.
- Validates department, class and subclass.

VALIDATE_LOCATION

- Only stockholding (virtual) warehouses can be on a stock count.

PROCESS_PROD

- Validates and creates a STAKE_PRODUCT record. No validation is done if the record is passed in for initial processing.

PROCESS_LOC

- Validates and creates a STAKE_LOCATION record. No validation is done if the record is passed in for initial processing.

PROCESS_DEL
CREATE_SH_REC

- Creates a record for STAKE_HEAD.

CREATE_SP_REC

- Creates a STAKE_PRODUCT record.

DELETE_RECS

- Deletes from STAKE_PRODUCT and STAKE_LOCATION tables.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
StkCountSchCre	Stock Count SCH Create Message	StkCountSchDesc.xsd
StkCountSchMod	Stock Count SCH Modify Message	StkCountSchDesc.xsd
StkCountSchDel	Stock Count SCH Delete Message	StkCountSchRef.xsd

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
DEPS	Yes	No	No	No
STORE	Yes	No	No	No
WH	Yes	No	No	No
STAKE_HEAD	Yes	Yes	Yes	No
STAKE_PRODUCT	No	Yes	No	Yes
STAKE_LOCATION	No	Yes	No	Yes
SYSTEM_OPTIONS	Yes	No	No	No

Store Subscription API

Functional Area

Foundation Data

Business Overview

The Store Subscription API provides the ability to keep store data in RMS in sync with an external system, if RMS is not being used as the system of record for organizational hierarchy information. The store data handled by the API includes basic store data plus relationship data between stores and their location traits and walk-through stores.

When creating a new store in RMS, the API uses RMS store creation batch logic. When a store creation message is received, it is validated and placed onto a staging table STORE_ADD. The store creation in RMS reads from this table and creates the store in RMS in an asynchronous mode.

When updating an existing store in RMS, the API performs the update immediately upon message receipt.

The API also handles store delete messages. But, like the store creation message subscription process, stores will not actually be deleted from the system upon receipt of the message. After the data has been validated, the store is added to the DAILY_PURGE table for processing via a batch process.

By default, stores inherit the location traits of the district to which they belong. However, specific location traits can also be assigned at the store level. Using the incoming external data, the API will create or delete relationships between stores and existing location traits.

Walkthrough stores are used in RMS as part of the transfer reconciliation process and are used to indicate two or more stores that have a 'walk through' connection between them – on the sales floor and/or the backroom. Using the incoming external data, the API will create or delete these relationships with stores as well.

Location trait and walkthrough store data cannot be sent in on a store create message. The store creates program must first process the store before it can have details attached to it.

Location trait and walkthrough store data must be processed separately as they each have their own distinct message types. These detail create messages will contain a snapshot of the store record. Note that location traits must already exist prior to being added to the store.

Deletion of location trait and walkthrough store relationships will also be handled within this API. The detail delete messages must be processed separately as they each have their own distinct message types.

Package Impact

Consume Module

Filename: rmssub_xstores/b.pls

```
RMSSUB_XSTORE.CONSUME
               (O_status_code        IN OUT        VARCHAR2,
                O_error_message      IN OUT        VARCHAR2,
                I_message            IN            RIB_OBJECT,
                I_message_type       IN            VARCHAR2)
```

This procedure will initially ensure that the passed in message type is a valid type for store messages. If the message type is invalid, a status of 'E' will 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 be downcast to the actual object using the Oracle's treat function. If the downcast fails, a status of 'E' will 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 will verify that the message passes all of RMS's business validation. It does not actually perform any validation itself, instead, it will call the RMSSUB_XSTORE_VALIDATE.CHECK_MESSAGE function to determine whether the message is valid. If the message has failed RMS business validation, a status of 'E' will 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 by calling RMSSUB_XSTORE_SQL.PERSIST_MESSAGE() function. 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_MESSAGE() function.

Once the message has been successfully persisted, 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_XSTORE.HANDLE_ERROR() – This is the standard error handling function that wraps the API_LIBRARY.HANDLE_ERROR function.

Business Validation Module

Filename: rmssub_xstorevals/b.pls

```
RMSSUB_XSTORE_VALIDATE.CHECK_MESSAGE
           (O_error_message   IN OUT         VARCHAR2,
            O_store_rec       OUT            NOCOPY  STORE_SQL.STORE_ROW_TYPE,
            I_message         IN             RIB_XStoreDesc,
            I_message_type    IN             VARCHAR2)
```

This function performs all business validation associated with messages and builds the store record for persistence. Some of the key validations performed are:

- Check if a like store was passed in. If it is, then the price store and cost location must match the like store. If a like store was not passed in, the copy replenishment, activity, and delivery indicators must be No or null.
- For new stores, check that the store number passed in is not currently being used for a store or warehouse. Note that stores and warehouses in RMS cannot have the same unique identifier.
- Verify the start order days are greater than or equal to zero.
- For updates or deletes, verify the store exists on the base table.

Bulk or Single DML Module

All insert, update and delete SQL statements are located in the family package. This package is STORE_SQL. The private functions in RMSSUB_STORE_SQL will call this package.

Filename: rmssub_xstoresqls/b.pls

```
RMSSUB_XSTORE_SQL.PERSIST_MESSAGE
               (O_error_message       IN OUT         VARCHAR2,
                I_store_rec           IN             STORE_SQL. STORE_ROW_TYPE,
                I_message_type        IN             VARCHAR2,)
```

This function determines what type of database transaction it will call based on the message type.

STORE CREATE

- Create messages get added to the staging table to be processed in a batch cycle. The address on the message is inserted as the primary address for the primary address type in the ADDR table. No other detail (child) processing occurs for creates.

STORE MODIFY

- Modify messages directly update the store table with changes. The address on the message is updated in the ADDR table. If the stores district has changed, the location traits from the old district will be removed, and the location traits for the new district will be added.

LOCATION TRAIT CREATE

- Adds location trait(s) to the store.

WALKTHROUGH CREATE

- Adds walkthrough store(s) to the store.

LOCATION TRAIT DELETE

- Removes location trait(s) to the store.

WALKTHROUGH DELETE

- Removes walkthrough store(s) to the store.

STORE DELETE

- Store gets added to a purging table to be processed in a batch cycle.

Message XSD

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 Type	Message Type Description	XML Schema Definition (XSD)
XStoreCre	External Store Create	XStoreDesc.xsd
XStoreDel	External Store Delete	XStoreRef.xsd
XStoreLocTrtCre	External Store Location Trait Create	XStoreDesc.xsd
XStoreLocTrtDel	External Store Location Trait Delete	XStoreRef.xsd
XStoreMod	External Store Modification	XStoreDesc.xsd
XStoreWTCre	External Walk Through Store Create	XStoreDesc.xsd
XStoreWTDel	External Walk Through Store Delete	XStoreRef.xsd

Design Assumptions

- Location traits already exist in RMS.
- Location trait and walkthrough store data cannot be sent in on a store create message.
- Some of the business validation is referential or involves uniqueness. This validation is handled automatically by the referential integrity constraints and the unique indexes implemented on the database.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
STORE_ADD	No	Yes	No	No
STORE	Yes	No	Yes	No
ADDR	Yes	Yes	Yes	No
DAILY_PURGE	No	Yes	No	No
LOC_TRAITS_MATRIX	Yes	Yes	No	Yes
SYSTEM_OPTIONS	Yes	No	No	No
TSF_ENTITY	Yes	No	No	No
WH	Yes	No	No	No
WALK_THROUGH_STORE	No	Yes	No	Yes
LOC_DISTRICT_TRAITS	Yes	No	No	No

Transfer Subscription API

Functional Area

Transfer

Business Overview

RMS subscribes to transfers from external systems to create, update or delete transfers in RMS. Oracle Retail's Advanced Inventory Planning system (AIP) also utilizes this API to create standalone warehouse to warehouse and warehouse to store transfers.

The transfer RIB API has defaulting logic which the API uses to populated defaulted fields. This is designed so that multiple sources can use the transfer API without having to conform to the same default values. Retailers can set-up their own set of default values or logic without having to modify the API code. For fields that are exposed on the message, if a value is provided, it will be used. Default values will only be used if a value is not provided on the message.

L10N Localization Decoupling Layer:

This is a layer of code which enables decoupling of localization logic that is only required for certain country-specific configuration. This layer affects the RIB API flows including Transfer subscription. This allows RMS to be installed without requiring customers to install or use this localization functionality, where not required.

Package Impact

Filename: rmssub_xtsfs/b.pls

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

This procedure initially ensures that the passed in message type is a valid type for transfer messages.

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

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

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

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

Filename: rmssub_xtsfvals/b.pls

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

RMSSUB_XTSF_VALIDATE.CHECK_MESSAGE

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

Like other APIs, the transfer API expects a snapshot of the record on both a header modify and a detail modify message, instead of only the fields that are changed. For a detail creates or a detail modify message, only the TSF number is validated at the header level; all other header fields are ignored.

TRANSFER CREATE

- Checks required fields.
- Validates fields.
- Defaults fields (status at header, freight type and tsf type).
- Builds transfer records.

TRANSFER MODIFY

- Checks required fields on the header nodes.
- Verifies TSF number already exists.
- Validates fields.
- Populates record.

TRANSFER DETAIL CREATE

- Checks required fields on the detail node.
- Verifies TSF number already exists.
- Verifies tsf/item/loc does **not** already exist.
- Creates item/loc relation if not already exists, including creating ITEM_LOC_SOH, ITEM_SUPP_COUNTRY_LOC, and PRICE_HIST records. If a pack item is involved, these records are created for all component items.
- Populates record.

TRANSFER DETAIL MODIFY

- Checks required fields on the detail node.
- Verifies transfer/item/loc already exists.
- If TSF quantity is reduced, verifies the new quantity is not below what has already been received plus what is being shipped or expected.
- Populates record.

RMSSUB_XTSF_VALIDATE.CHECK_MESSAGE

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

TRANSFER DELETE

- Checks required fields.
- Verifies TSF number already exists.
- Verifies that TSF is not already shipped or received.
- Populates record for delete.

TRANSFER DETAIL DELETE

- Checks required fields.
- Verifies TSF/item/loc already exists.
- Verifies that TSF line is not already shipped or received.
- Populates record with the TSF no/item/location for delete.

Filename: rmssub_xtsfs/b.pls

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

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

TRANSFER CREATE

- Inserts records in the TSFHEAD, TSFDETAIL, TSFDETAIL_CHRG tables.
- Updates records in the ITEM_LOC_SOH table.

TRANSFER MODIFY

- Updates a record in the TSFHEAD table.

TRANSFER DELETE

- Deletes a transfer from TSFHEAD, TSFDETAIL, TSFDETAIL_CHRG tables.

TRANSFER DETAIL CREATE

- Inserts records in the TSFDETAIL, TSFDETAIL_CHRG tables.
- Updates records in the ITEM_LOC_SOH table.

TRANSFER DETAIL MODIFY

- Updates records in the TSFDETAIL, ITEM_LOC_SOH tables.

TRANSFER DETAIL DELETE

- Delete records from TSFDETAIL, TSFDETAIL_CHRG tables.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
Xtsfcre	Transfer Create Message	XTsfDesc.xsd
Xtsfmod	Transfer Modify Message	XTsfDesc.xsd
Xtsfdel	Transfer Delete Message	XTsfRef.xsd
Xtsfdtlcre	Transfer Detail Create Message	XTsfDesc.xsd
Xtsfdtlmod	Transfer Detail Modify Message	XTsfDesc.xsd
Xtsfdtlcel	Transfer Detail Delete Message	XTsfRef.xsd

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
TSFHEAD	Yes	Yes	Yes	Yes
TSFDETAIL	Yes	Yes	Yes	Yes
TSFDETAIL_CHRG	Yes	Yes	Yes	Yes
ITEM_LOC	Yes	Yes	No	No
ITEM_LOC_SOH	Yes	Yes	No	No
PRICE_HIST	No	Yes	No	No
ITEM_MASTER	Yes	No	No	No
PACKITEM_BREAKOUT	Yes	No	No	No
STORE	Yes	No	No	No
WH	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No
WF_ORDER_HEAD	Yes	Yes	Yes	No
WF_ORDER_DETAIL	Yes	Yes	Yes	Yes
WF_ORDER_EXP	Yes	Yes	Yes	Yes

Vendor Subscription API

Functional Area

Foundation Data

Business Overview

RMS subscribes to supplier information that is published from an external financial application. 'Vendor' refers to either a partner or a supplier, but only supplier information is subscribed to by RMS. Supplier information also includes supplier addresses and the org unit.

Processing includes a check for the appropriate financial application in RMS on the SYSTEM_OPTIONS table's FINANCIAL_AP column, which will result in different processing. The financial application (such as Oracle EBS) sends the information to RMS through RIB.

The financial application publishes a supplier type vendor, placing the supplier information onto the RIB (Oracle Retail Information Bus). RMS subscribes to the supplier information as published from the RIB and places the information onto RMS tables depending upon the validity of the records enclosed within the message.

Package Impact

Filename: rmssub_vendorcres/b.pls

Public API Procedures

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

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

Private Internal Functions and Procedures (rmssub_vendorcre.pls):

Error Handling

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

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

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

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

Private Internal Functions and Procedures (other):

All of the following functions exist within RMSSUB_SUPPLIER.

Main Consume Function

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

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

The record is processed and then validates the XML file format and, if successful, calls internal functions to parse the values within the file through a series of calls to RIB_XML. The values extracted from the file are then passed on to private internal functions, which validate the values and place them on the appropriate supplier and address database tables depending upon the success of the validation. The procedure then calls the PROCESS_ADDRESS function to check that the proper addresses have been associated with the supplier and store the address details in ADDR table. After processing the address records, the procedure calls PROCESS_ORGUNIT function to process the org units.

PARSE_SUPPLIER

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

The record is based upon the supplier table.

PARSE_ADDRESS

This function extracts the address level information from the supplier XML file and places that information onto an internal address record.

The record is based upon the address table.

PROCESS_SUPPLIER

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

PROCESS_ADDRESS

After the values are parsed for a particular address record, RMSSUB_SUPPLIER.CONSUME calls this function. If the FINANCIAL_AP system option is set to '0', this function calls various functions inside RMSSUB_SUPPLIER in

order to validate the values and place them on the appropriate address table depending upon the success of the validation. Either INSERT_ADDRESS or UPDATE_ADDRESS is called to actually insert or update the address table.

INSERT_SUPPLIER

This function first checks the PROCUREMENT_UNIT_OPTIONS table to determine what the value of dept_level_orders is. If the dept_level_orders value is 'Y', the inv_mgmt_lvl is defaulted to 'D'. If the dept_level_orders value is anything other than 'Y', the inv_mgmt_lvl is set to 'S.'

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

FUNCTION UPDATE_SUPPLIER

This function updates the SUPS table using the values contained in the I_supplier_record.

If the primary address of the supplier is localized then supplier status will be 'I' - Inactive.

FUNCTION UPDATE_ADDRESS

This function updates the supplier information to the address table.

CHECK_CODES

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

INSERT_ADDRESS

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

VALIDATE_SUPPLIER_RECORD

Validate that all the necessary records are populated. In the supplier site enabled environment (system_options.supplier_site_ind = 'Y') supplier_parent must be present.

VALIDATE_ADDRESS_RECORD

Validate that all the necessary records are populated.

CHECK_NULLS

This function checks that the passed-in record variable is not null. If it is, it will return an error message.

VALIDATE_ORG_UNIT_RECORD

This function checks that the passed-in record variable is not null. If it is, it will return an error message. When not null, it checks for a valid org unit in ORG_UNIT table.

PROCESS_ORGUNIT

After validating the org unit, this function either inserts or updates the record in PARTNER_ORG_UNIT table. If the vendor/orgunit in the passed-in Org Unit record is the primary pay site for a particular vendor/orgunit type, this function updates the current primary paysite so that it is no longer the primary. When supplier_site_ind = 'Y', partner_org_unit only exists for supplier sites, not for parent supplier hence this function will be called for supplier sites and not for supplier.

Message XSD

Here are the filenames that correspond with each message type. Please consult Oracle Retail Integration Bus information for each message type in order to get a detailed picture of the composition of each message.

Message Types	Message Type Description	XML Schema Definition (XSD)
VendorCre	Vendor Create Message	VendorDesc.xsd

Design Assumptions

NA

Table Impact

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

Work Order Status Subscription API

Functional Area

Work Order Status

Business Overview

RMS subscribes to a work order status message sent from internal finishers. Work order status messages contain the items for which the activities have been completed along with the quantity that was completed. All items on transfers that pass through an internal finisher must have at least one work order activity performed upon them. When work order status messages are received for a particular item/quantity, it is assumed that all work order activities associated with the item/quantity have been completed. If work order activities involve item transformation or repacking, the work order status messages are always created in terms of the resultant item.

The work order status message is only necessary when the internal finisher and the final receiving location are in the same physical warehouse. If the internal finisher belongs to the receiving location, a book transfer is made between the internal finisher (which is held as a virtual warehouse) and the final receiving location (also a virtual warehouse). If the internal finisher belongs to the sending location's transfer entity, intercompany out and intercompany in transactions are recorded. Quantities on hand, reserved quantities, and weighted average costs are adjusted to accurately reflect the status of the stock.

Assume that a quantity of 20 of item 100 (White XL T-shirt) are sent to an internal finisher at the receiving physical warehouse where they will be dyed black, thereby transforming them into item 101 (Black XL T-shirt). If all finishing activities were successfully completed in this example, RMS could expect to receive a Work Order Status message containing item 101 with a quantity of 20.

It is possible to receive multiple Work Order Status messages for a particular item/transfer. Work order completion of partial quantities addresses the following scenarios:

1. Work order activities could not be performed for the entire quantity of a particular item at one time.
2. A given quantity of the particular item was damaged while work order activities were performed.

In terms of the previous example, RMS could receive a message containing item 101(Black XL T-shirt) with a quantity of 10. A message stating that work order activities were completed for the remaining 10 items could then be received at a later time.

The only scenario in which a Work Order Status message is necessary is when work order activities are taking place at an internal finisher that resides in the same physical warehouse as the transfer's final receiving location. In this scenario, the final 'leg' of the transfer will 'move' merchandise between two virtual warehouses in the same physical warehouse. As this movement cannot be done until all work order activities are completed for a specific item/quantity, the finisher must inform RMS of this completion.

Other finishing scenarios exist in which the finisher is not a virtual warehouse that shares a physical warehouse with the transfer's final receiving location. In these instances, Work Order Status messages are not necessary. This is because these scenarios dictate that merchandise must be physically shipped from the finisher to the transfer's final receiving location. RMS assumes that a finisher will not ship merchandise until all finishing activities have been completed for said merchandise. RMS will disregard Work Order Status messages sent in these scenarios.

Package Impact

Filename: rmssub_wostatuss/b.pls

```
PROCEDURE CONSUME
              (O_status_code         IN OUT         VARCHAR2,
               O_error_message       IN OUT         VARCHAR2,
               I_message             IN             RIB_OBJECT,
               I_message_type        IN             VARCHAR2)
```

This procedure is passed an Oracle Object, which it will validate to ensure all required data is present. It will ensure that the finisher and the transfer's final receiving location are in the same physical warehouse. If not, processing is deemed successful and halted. If the message contains an item, RMS work order complete processing will be called for that item. Otherwise, said processing will be called for all items on the transfer. If the entire transfer is processed, the child transfer (that is, the 'second leg') will be set to 'S'hipped status. Note that work orders are always associated with the second leg of multi-leg transfers. Whether processing is performed at the item or transfer level, transfer closing queue logic will be called to determine if the entire multi-leg transfer can be closed.

```
PROCEDURE HANDLE_ERRORS
              (O_status_code         IN OUT         VARCHAR2,
               IO_error_message      IN OUT         VARCHAR2,
               I_cause               IN             VARCHAR2,
               I_program             IN             VARCHAR2)
```

This is the standard error handling procedure that wraps the API_LIBRARY.HANDLE_ERROR function.

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
wostatuscre	Work Order Status Create Message	WOStatusDesc.xsd

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
TSFHEAD	Yes	No	Yes	No
TSF_DETAIL	Yes	No	Yes	No
TSF_ITEM_COST	Yes	No	Yes	No
DOC_CLOSE_QUEUE	No	Yes	No	No
ITEM_LOC_SOH	Yes	Yes	Yes	No
TRAN_DATA(VIEW)	No	Yes	No	No
INV_ADJ	No	Yes	No	No
INV_STATUS_QTY	No	Yes	Yes	Yes
INV_ADJ_REASON	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
INV_STATUS_CODES	Yes	No	No	No
SHIPSKU	Yes	No	No	No

4

Web Service Provider Implementation

This chapter gives an overview about the Web service provider implementation API designs used in the RMS environment and various functional attributes used in the APIs.

> **Note:** The following service provider implementation API designs are intended only to give a high level overview of the APIs available.
>
> The implementation of these services, along with the associated Web Service Definition Language (WSDL), may be used to get a full understanding of the data requirements, validation rules, persistence rules, and return values associated with the service.

Supplier Service

Functional Area

Foundation Data

Business Overview

RMS subscribes supplier information from external financial applications via this Web service. The Supplier Service Provider is used by external financial systems to send RMS the supplier information that includes supplier addresses and the operating unit. The header record contains information about the supplier as a whole. The address records identify the addresses associated with the supplier and the operating unit records specify the operating units associated with the supplier.

Package Impact

This public package is called by the supplier Web service to send supplier information to RMS.

Filename: rmsaiasub_suppwebss/b.pls

Public API Procedures

```
SupplierServiceProviderImpl.createSupplierDesc
                  (I_serviceoperationcontext  IN   "RIB_ServiceOpContext_REC",
                   I_businessobject           IN   "RIB_SupplierDesc_REC",
                   O_serviceoperationstatus   OUT  "RIB_ServiceOpStatus_REC",
                   O_businessobject           OUT  "RIB_SupplierRef_REC")
```

This procedure populates the first record of "RIB_SupplierColDesc_REC" and passes the record to the function RMSAIASUB_SUPPLIER.CONSUME() with a message type of 'suppadd'. The procedure RMSAIA_LIB.BUILD_SERVICE_OP_STATUS() is used to return status to the calling Web service. If there is any error then the O_error_message from consume will be assigned to the RIB_OBJECT O_serviceOperationStatus as per the signature of the new RIB_OBJECT.

```
SupplierServiceProviderImpl.updateSupplierDesc
                  (I_serviceoperationcontext  IN   "RIB_ServiceOpContext_REC",
                   I_businessobject           IN   "RIB_SupplierDesc_REC",
                   O_serviceoperationstatus   OUT  "RIB_ServiceOpStatus_REC",
                   O_businessobject           OUT  "RIB_SupplierDesc_REC")
```

This procedure populates the first record of "RIB_SupplierColDesc_REC" and passes the record to the function RMSAIASUB_SUPPLIER.CONSUME() with a message type of 'suppmod'. The procedure RMSAIA_LIB.BUILD_SERVICE_OP_STATUS() is used to return status to the calling Web service. If there is any error then the O_error_message from consume will be assigned to the RIB_OBJECT O_serviceOperationStatus as per the signature of the new RIB_OBJECT.

The following procedures are part of the package, but are not supported by RMS. When called by the Web service, these procedures will return without further processing with an error message "This webservice is not supported now". These procedures are:

- SupplierServiceProviderImpl.createSupSiteUsingSupplierDesc()
- SupplierServiceProviderImpl.updateSupSiteUsingSupplierDesc()
- updateSupSiteAddrUsingSupplier()
- updateSupSiteOrgUnitUsingSuppl()
- updateSupSiteUsingSupplierDesc()
- createSupSiteAddrUsingSupplier()
- createSupSiteUsingSupplierDesc()
- findSupplierDesc()
- deleteSupplierDesc()
- findSupplierColDesc()
- deleteSupplierColDesc()

```
SupplierServiceProviderImpl.updateSupplierColDesc
                  (I_serviceOperationContext IN      "RIB_ServiceOpContext_REC",
                   I_businessObject          IN      "RIB_SupplierColDesc_REC",
                   O_serviceOperationStatus     OUT  "RIB_ServiceOpStatus_REC",
                   O_businessObject             OUT  "RIB_SupplierColDesc_REC")
```

This procedure passes the record to the function RMSAIASUB_SUPPLIER.CONSUME() with message type as "suppmod". The procedure RMSAIA_LIB.BUILD_SERVICE_OP_STATUS() is used to return status to the calling Web service.

```
SupplierServiceProviderImpl.createSupplierColDesc
           (I_serviceoperationcontext    IN   "RIB_ServiceOpContext_REC",
            I_businessobject             IN   "RIB_SupplierColDesc_REC",
            O_serviceoperationstatus     OUT  "RIB_ServiceOpStatus_REC",
            O_businessobject             OUT  "RIB_SupplierColRef_REC")
```

This procedure passes the record to the function RMSAIASUB_SUPPLIER.CONSUME() with message type as "suppadd". The procedure RMSAIA_LIB.BUILD_SERVICE_OP_STATUS() is used to return status to the calling Web service.

Private Internal Functions and Procedures (rmsaiasub_supplierb/s.pls)

This is the main consume function:

```
RMSAIASUB_SUPPLIER.CONSUME
                (O_status_code        IN OUT   VARCHAR2,
                 O_error_message      IN OUT   RTK_ERRORS.RTK_TEXT%TYPE,
                 O_outputobject       IN OUT   "RIB_SupplierColRef_REC",
                 I_inputobject        IN       "RIB_SupplierColDesc_REC",
                 I_inputobject_type   IN       VARCHAR2)
```

This procedure is called by the package `SupplierServiceProviderImpl` to consume supplier information coming from the Financial System via the Web service. It then validates the data and persists it to the RMS Supplier tables. It does most of the validation through the RMSAIASUB_SUPPLIER_VALIDATE.PROCESS_SUPPLIER_RECORD() function, which utilizes the internal functions VALIDATE_RECORD() and POPULATE_RECORD. After the validation checks the data, the RMSAIASUB_SUPPLIER_SQL.PERSIST() is called to update the RMS supplier maintenance tables.

Private Internal Functions and Procedures (rmsaiasub_supplier_valb/s.pls)

```
RMSSUB_SUPPLIER_VAL.PROCESS_SUPPLIER_RECORD
                          (O_error_message     IN OUT  RTK_ERRORS.RTK_TEXT%TYPE,
                           O_supplier_object   IN OUT  SUPP_REC,
                           O_ref_outputobject  IN OUT  "RIB_SupplierColRef_REC",
                           I_inputobject       IN      "RIB_SupplierColDesc_REC",
                           I_inputobject_type  IN      VARCHAR2)
```

This function is the main validation function for the supplier Web service interface. It calls various internal functions to verify NULLs, validate codes in the CODE_DETAIL table, or confirm values have the necessary foreign keys in the RMS system. If the validation processes do not fail, the next step is to populate the local record groups to be used later for populating RMS tables.

Private Internal Functions and Procedures (rmsaiasub_supplier_sqlb/s.pls)

```
RMSAIASUB_SUPPLIER_SQL.PERSIST
               (O_error_message      IN OUT   RTK_ERRORS.RTK_TEXT%TYPE,
                I_supplier_record    IN OUT   SUPP_REC)
```

This function is called from RMSAIASUB_SUPPLIER.CONSUME() to insert into the RMS tables. The following internal functions are utilized:

INSERT_SUPPLIER

This function will insert to the SUPS table is it does not exist yet, and will update the records if it already exists in RMS.

INSERT_SUPPLIER_SITES

This function will insert to the SUPS table is it does not exist yet, and will update the records if it already exists in RMS.

INSERT_ADDRESS

This function will insert to the ADDR table is it does not exist yet, and will update the records if it already exists in RMS.

INSERT_ORG_UNIT

This function will insert to the PARTNER_ORG_UNIT table is it does not exist yet, and will update the records if it already exists in RMS.

Design Assumptions

- The Web service initially calls the package SupplierServiceProviderImpl that serves as a wrapper for the main consume function RMSAIASUB_SUPPLIER.CONSUME(). The consume function utilizes the packages RMSSUB_SUPPLIER_VAL and RMSAIASUB_SUPPLIER_SQL to process the supplier data consumption.

Table Impact

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

Pay Term Service

Functional Area

Financial Integration

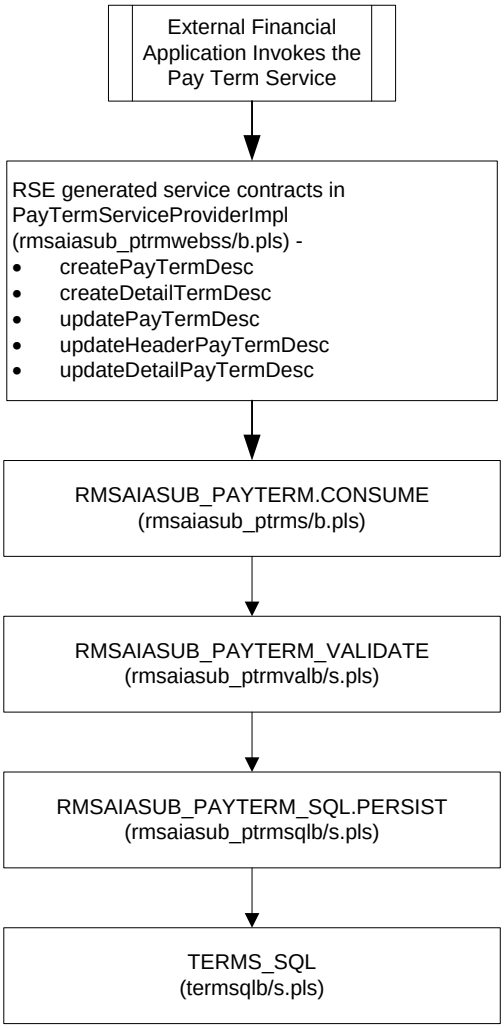
Business Overview

The Pay Term Service Provider is used by external financial systems to send RMS new and updated payment term information. Header and detail level payment term information is written to the TERMS_HEAD and TERMS_DETAIL tables.

This document describes the Pay Term Web service integration between RMS and an external financial application. In this integration context, RMS acts as the service provider that exposes a Web service to be invoked by an external financial application.

Package Impact

The process flow diagram for Pay Term Service API.



Process Flow for Pay Term Service

Public API Procedures

Filename: rmsaiasub_ptrmwebss/b.pls

Package name: PayTermServiceProviderImpl

```
createPayTermDesc
      (I_serviceOperationContext  IN      "RIB_ServiceOpContext_REC",
       I_businessObject           IN      "RIB_PayTermDesc_REC",
       O_serviceOperationStatus       OUT  "RIB_ServiceOpStatus_REC",
       O_businessObject               OUT  "RIB_PayTermRef_REC")
```

This procedure corresponds to the 'create' operation of the Pay Term Web service. It calls RMSAIASUB_PAYTERM.CONSUME with a message type of RMSAIASUB_PAYTERM.HDR_ADD to create payment terms in RMS. It returns RMS's pay term ID through output O_businessObject and success or failure status through O_serviceOperationStatus.

```
createDetailPayTermDesc
       (I_serviceOperationContext  IN      "RIB_ServiceOpContext_REC",
        I_businessObject           IN      "RIB_PayTermDesc_REC",
        O_serviceOperationStatus         OUT  "RIB_ServiceOpStatus_REC",
        O_businessObject                 OUT  "RIB_PayTermRef_REC")
```

This procedure corresponds to the 'createDetail' operation of the Pay Term Web service. It calls RMSAIASUB_PAYTERM.CONSUME with a message type of RMSAIASUB_PAYTERM.DTL_ADD to create payment term details in RMS. It returns RMS's pay term ID through output O_businessObject and success or failure status through O_serviceOperationStatus.

```
updatePayTermDesc
       (I_serviceOperationContext  IN      "RIB_ServiceOpContext_REC",
        I_businessObject           IN      "RIB_PayTermDesc_REC",
        O_serviceOperationStatus      OUT  "RIB_ServiceOpStatus_REC",
        O_businessObject              OUT  "RIB_PayTermDesc_REC")
```

This procedure corresponds to the 'update' operation of the Pay Term Web service. It calls RMSAIASUB_PAYTERM.CONSUME with a message type of RMSAIASUB_PAYTERM.DTL_UPD to update payment terms in RMS. It returns a "RIB_PayTermDesc_REC' object through output O_businessObject and success or failure status through O_serviceOperationStatus.

```
updateHeaderPayTermDesc
       (I_serviceOperationContext  IN      "RIB_ServiceOpContext_REC",
        I_businessObject           IN      "RIB_PayTermDesc_REC",
        O_serviceOperationStatus      OUT  "RIB_ServiceOpStatus_REC",
        O_businessObject              OUT  "RIB_PayTermDesc_REC")
```

This procedure corresponds to the 'updateHeader' operation of the Pay Term Web service. It calls RMSAIASUB_PAYTERM.CONSUME with a message type of RMSAIASUB_PAYTERM.HDR_UPD to update header level payment term information in RMS. It returns a "RIB_PayTermDesc_REC' object through output O_businessObject and success or failure status through O_serviceOperationStatus.

```
updateDetailPayTermDesc
       (I_serviceOperationContext  IN      "RIB_ServiceOpContext_REC",
        I_businessObject           IN      "RIB_PayTermDesc_REC",
        O_serviceOperationStatus      OUT  "RIB_ServiceOpStatus_REC",
        O_businessObject              OUT  "RIB_PayTermDesc_REC")
```

This procedure corresponds to the 'updateDetail' operation of the Pay Term Web service. It calls RMSAIASUB_PAYTERM.CONSUME with a message type of RMSAIASUB_PAYTERM.DTL_UPD to update detail level payment term information in RMS. It returns a "RIB_PayTermDesc_REC' object through output O_businessObject and success or failure status through O_serviceOperationStatus.

Filename: rmsaiasub_ptrms/b.pls

Package name: RMSAIASUB_PAYTERM

```
CONSUME(O_status_code            OUT  VARCHAR2,
        O_error_message          OUT  VARCHAR2,
        O_rib_paytermref_rec     OUT  "RIB_PayTermRef_REC",
        I_message             IN      "RIB_PayTermDesc_REC",
        I_message_type        IN      VARCHAR2)
```

This procedure validates the message content in I_message with respect to the message type (I_message_type) and persists payment terms information in RMS's TERMS_HEAD and TERMS_DETAIL tables. It returns RMS's term ID through output O_rib_paytermref_rec.

Filename: rmsaiasub_ptrmvals/b.pls

Package name: RMSAIASUB_PAYTERM_VALIDATE

```
FUNCTION CHECK_MESSAGE(O_error_message  IN OUT  RTK_ERRORS.RTK_TEXT%TYPE,
                       O_paytermref_rec IN OUT  "RIB_PayTermRef_REC",
                       O_payterm_rec    IN OUT  TERMS_SQL.PAYTERM_REC,
                       I_message        IN      "RIB_PayTermDesc_REC",
                       I_message_type   IN      VARCHAR2)
RETURN BOOLEAN;
```

This function performs validation on the message content in I_message with respect to the message type (I_message_type). It returns RMS's term ID through output O_rib_paytermref_rec.

Filename: rmsaiasub_ptrmsqls/b.pls

Package name: RMSAIASUB_PAYTERM_SQL

```
FUNCTION PERSIST(O_error_message     OUT  RTK_ERRORS.RTK_TEXT%TYPE,
                 I_message        IN      TERMS_SQL.PAYTERM_REC,
                 I_message_type   IN      VARCHAR2)
RETURN BOOLEAN;
```

This function calls TERMS_SQL.MERGE_HEADER and TERMS_SQL.MERGE_DETAIL functions to persist payment terms information to RMS's TERMS_HEAD and TERMS_DETAIL tables.

Filename: termsqls/b.pls

Package name: TERMS_SQL

```
FUNCTION MERGE_HEADER(O_error_message IN OUT RTK_ERRORS.RTK_TEXT%TYPE,
                      I_paytermrec    IN     PAYTERM_REC)
RETURN BOOLEAN;
```

This function persists payment terms header level information to RMS's TERMS_HEAD table.

```
FUNCTION MERGE_DETAIL(O_error_message IN OUT RTK_ERRORS.RTK_TEXT%TYPE,
                      I_paytermrec    IN     PAYTERM_REC)
RETURN BOOLEAN;
```

This function persists payment terms detail level information to RMS's TERMS_DETAIL table.

Message XSD

Message Types	Message Type Description	XML Schema Definition (XSD)
paytermcre	create payment terms	PayTermDesc.xsd, PayTermRef.xsd
paytermmod	update payment terms header	PayTermDesc.xsd, PayTermRef.xsd
paytermdtlcre	create payment terms detail	PayTermDesc.xsd, PayTermRef.xsd
paytermdtlmod	Update payment terms detail	PayTermDesc.xsd, PayTermRef.xsd

Design Assumptions

NA

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
SYSTEM_OPTIONS	Yes	No	No	No
TERMS_HEAD	Yes	Yes	Yes	No
TERMS_DETAIL	Yes	Yes	Yes	No

Customer Order Fulfillment Service

Functional Area

Customer Order Fulfillment

Business Overview

RMS provides an interface to process Customer Order Fulfillment requests from an external order management system (OMS). If the system option OMS_IND = 'Y', then RMS expects to receive customer orders via this API.

RMS supports two integration methods for processing Customer Order Fulfillment messages from OMS – either through RIB or Web service. At implementation time, clients should decide on either one or the other integration method, but not both. The same core logic is used to validate and persist customer orders to RMS tables.

- In a RIB implementation, RMS subscribes to Customer Order Fulfillment messages. When a customer order is created, or partially or fully cancelled, the customer order information is sent from the Order Management System (OMS) to the RIB. RMS subscribes to the customer order information as published from the RIB and places the information onto RMS tables.
- In a Web service implementation, RMS exposes a FulfillOrder Web service to create or cancel a customer order in RMS. OMS will invoke the service with customer order details to place the information on RMS tables.

The Customer Order Fulfillment message staged in the RMS tables will go through a process of validation. Records that pass validation will create new customer order

records. If any validation error occurs, transaction will be rolled back and no customer orders will be created.

There are two scenarios where a customer order fulfillment request cannot be created in RMS:

1) Due to data validation errors (for example, invalid item).
2) Due to 'No Inventory' - There is not enough inventory available at the source location or item is not ranged or inactive at the source location, or item is not supplied by the supplier (in a PO scenario).

Web Service Deployment

- Accepts a collection of fulfillment orders as input. If one order fails, the entire service call fails and no order will be created in RMS.
- RMS returns Failure status as part of the response object in the Web service call if customer orders are not created due to validation errors.
- RMS returns Success status and a confirmation message of type 'X' as part of the response object if customer orders are not created due to 'No Inventory' or a confirmation message of type 'P' if customer orders are partially created due to insufficient inventory.

RIB Deployment

- Accepts a single fulfillment order as input to allow RIB's sequencing mechanism to work as designed.
- RMS returns Failure to the RIB and the message will land in the RIB hospital if a customer order is not created due to validation errors. No confirmation message will be sent.
- RMS returns Success. In a separate transaction, a confirmation message of type 'X' will be sent to the RIB if a customer order is not created due to 'No Inventory', or a confirmation message of type 'P' will be sent to the RIB if a customer order is partially created due to insufficient inventory. Based on the confirmation message, OMS will take action to source the order from a different location. See Customer Order Fulfillment Confirmation Publication API.

The Customer Order Fulfillment messages contain information such as delivery type, source type and destination type. Based on these, the system should proceed to create a Purchase Order, Transfer or Inventory Reservation. The table below shows the customer order scenarios for the combination of delivery type, source type and destination type.

Scenario #	Source Location	Fulfillment Location	Delivery Type	Transaction created
1	Warehouse	Store	Pickup in Store	Virtual WH to Physical Store Transfer + Reservation. FulfilOrdDesc will contain: 1st leg: source_loc_type = 'WH', fulfill_loc_type = 'S' 2nd leg: source_loc_type = NULL, fulfill_loc_type = 'S'.

Scenario #	Source Location	Fulfillment Location	Delivery Type	Transaction created
2	Warehouse	Store	Ship to Customer	Virtual WH to Physical Store Transfer + Reservation. FulfilOrdDesc will contain: 1st leg: source_loc_type = 'WH', fulfill_loc_type = 'S' 2nd leg: source_loc_type = NULL, fulfill_loc_type = 'S'.
3	Store A	Store B	Pickup in Store	Physical Store to Physical Store Transfer + Reservation. FulfilOrdDesc will contain: 1st leg: source_loc_type = 'ST', fulfill_loc_type = 'S' 2nd leg: source_loc_type = NULL, fulfill_loc_type = 'S'.
4	Store A	Store B	Ship to Customer	Physical Store to Physical Store Transfer + Reservation. FulfilOrdDesc will contain: 1st leg: source_loc_type = 'ST', fulfill_loc_type = 'S' 2nd leg: source_loc_type = NULL, fulfill_loc_type = 'S'.
5	NULL	Store	Pickup in Store	Reservation. FulfilOrdDesc will contain: Single-leg: source_loc_type = NULL, fulfill_loc_type = 'S'.
6	NULL	Store	Ship to Customer	Reservation. FulfilOrdDesc will contain: Single-leg: source_loc_type = NULL, fulfill_loc_type = 'S'.
7	NULL	Warehouse	Ship to Customer	Virtual WH to Virtual Store Transfer. FulfilOrdDesc will contain: Single-leg: source_loc_type = 'WH', fulfill_loc_type = 'V'.
8	Vendor	Store	Pickup in Store	Purchase Order to Physical Store + Reservation. FulfilOrdDesc will contain: 1st leg: source_loc_type = 'SU', fulfill_loc_type = 'S' 2nd leg: source_loc_type = NULL, fulfill_loc_type = 'S'.
9	Vendor	Store	Ship to Customer	Purchase Order to Physical Store+ Reservation. FulfilOrdDesc will contain: 1st leg: source_loc_type = 'SU', fulfill_loc_type = 'S' 2nd leg: source_loc_type = NULL, fulfill_loc_type = 'S'.

Scenario #	Source Location	Fulfillment Location	Delivery Type	Transaction created
10	NULL	Vendor	Ship to Customer	Purchase Order to Virtual Store FulfilOrdDesc will contain: Single-leg: source_loc_type = 'SU', fulfill_loc_type = 'V'.

The customer order subscription API supports create and cancel operations using the following message types belonging to the 'fulfilord' message family:

- **fulfilordapprdel** – used by RMS to cancel customer orders.
- **fulfilordreqdel** – used by SIM to request a customer order cancellation. This message type is used only by SIM and is ignored by RMS.
- **fulfilordpocre** – used to create purchase orders as a result of customer order fulfillment requests.
- **fulfilordtsfcre** – used to create transfers as a result of customer order fulfillment requests.
- **fulfilordstdlvcre** – used to perform inventory reservation as a result of customer order fulfillment requests.

In a RIB implementation, once fulfillment create messages are processed in RMS, RMS will publish to the RIB a customer order fulfillment confirmation message with a message type of 'fulfilordcfmcre' via the customer order fulfillment confirmation publishing API, rmsmfm_ordcust. Confirmation messages will only be sent for customer order fulfillment creates requests that result in creating purchase orders and transfers in RMS. It will not be sent for cancel requests, or for customer order fulfillment requests that result in inventory reservation.

- If a customer order is partially fulfilled, a confirmation message with status 'P' will be sent with details of fulfilled order quantity.
- If a customer order is not fulfilled at all due to unavailable inventory, a confirmation message with status 'X' will be sent without any details.

In a Web service implementation, confirmation messages will be sent in a collection as part of the response object. In a RIB implementation, separate confirmation messages will be published from RMS in independent transactions.

Package Impact

Public Interface:

Filename: stgsvc_fulfilords/b.pls

This package provides public interfaces (pop_create_tables and pop_cancel_tables) to stage customer order fulfillment create and cancel requests in the collection to interface tables. It also provides a public interface (cleanup_tables) to clear out data in the interface staging tables after processing.

Business Validation Module

Filename: coresvc_fulfilordvals/b.pls

This package contains logic that performs generic validation of customer order fulfillment create and cancel requests in the following interface staging tables:

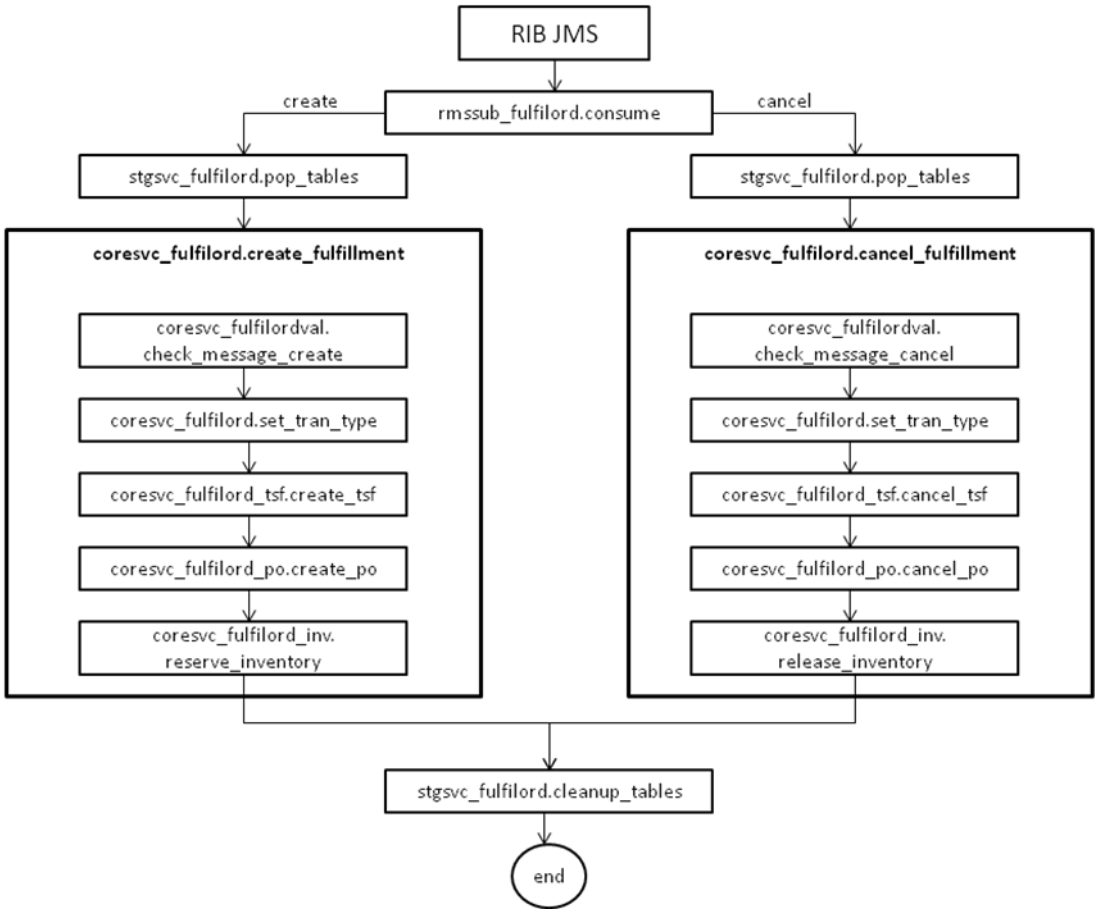
SVC_FULFILORD
SVC_FULFILORDDTL
SVC_FULFILORDCUST
SVC_FULFILORDREF
SVC_FULFILORDDTLREF

Subscription Package

Filename: rmssub_fulfilords/b.pls

RMS will subscribe to the customer order fulfillment create or cancel message from the RIB. The RIB message will be parsed and staged into staging tables for initial validation via stgsvc_fulfilord.pop_tables. The coresvc_fulfilordval package will be called to perform generic validation. If no error is encountered during initial validation, transfer, PO, inventory reservation specific validation functions will be invoked to perform further validation and to create customer order transfers, purchase orders, or reserve inventory in RMS. The staging table will be purged at the end of the processing.

The diagram RIB JMS Deployment for Customer Order Fulfillment Requests illustrates this process:



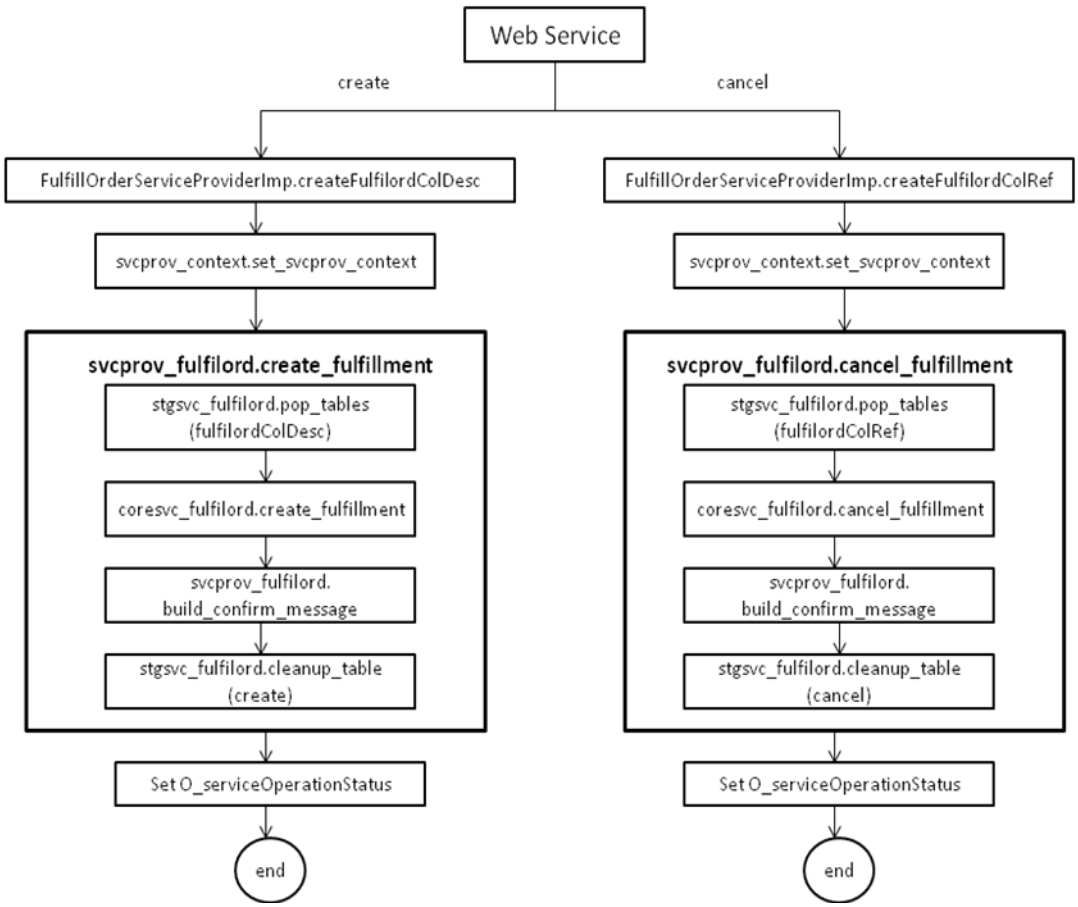
RIB JMS Deployment for Customer Order Fulfillment Requests

Filename: FulfillOrderServiceProviderImplSpec.pls FulfillOrderServiceProviderImplBody.pls

For a Web service deployment, a new Web service 'FulfillOrder' with two supported operations of 'create' and 'cancel' is available for OMS to send customer order fulfillment create and cancel requests to RMS. The Web services will invoke public interfaces for the Customer Order Fulfillment Create Request (FulfillOrderServiceProviderImp.createFulfilOrdColDesc) and the Customer Order Fulfillment Cancel Request (FulfillOrderServiceProviderImp.cancelFulfilOrdColRef).

These public interfaces calls create and cancel procedures in svcprov_fulfilord to do major processing logic. Similar to a RIB JMS deployment, the messages will be staged, validated, and persisted to RMS using the same core functions. At the end of the processing, the staging tables are purged and a confirmation status is returned.

The diagram Web Service Deployment for Customer Order Fulfillment Requests illustrates this process.



Web Service Deployment for Customer Order Fulfillment Requests

Message XSD

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

Message Types	Message Type Description	XML Schema Definition (XSD)
Fulfilordapprdel	Fulfilment Cancel Message	FulfilOrdRef.xsd
Fulfilordreqdel	Fulfilment Cancel Request Message	FulfilOrdRef.xsd
Fulfilordpocre	Fulfilment PO Create Message	FulfilOrdDesc.xsd
Fulfilordtsfcre	Fulfilment Transfer Create Message	FulfilOrdDesc.xsd
Fulfilordstdlvcre	Fulfilment Store Delivery Create Message	FulfilOrdDesc.xsd

Design Assumptions

1. Customer order fulfillment request cannot be created in RMS for the following scenarios:
   - Customer orders are not created due to any validation error.
   - Customer orders are created in 'X' status due to 'no inventory' (for example, not enough available at the source location, or item not ranged to or active at the source location, or in a PO scenario, item not supplied by the supplier).
2. Non-stockholding franchise stores cannot part of a customer order, either as a sourcing location or as a fulfillment location.
3. Only approved, inventoried and sellable items will be published to OMS. Therefore, item types like catch weight, concession, consignment, and transformable sellable items will NOT be published to OMS, and will NOT be supported by this interface. To sell items that can vary by weight, like bananas, through online channels, setup should be done as a regular (non-catch weight) item with a unit cost and standard UOM defined in items of eaches.
4. It is assumed that customer orders will be captured in the selling UOM in OMS, but that all transactions will be communicated to RMS in standard UOM.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
SVC_FULFILORDREF	Yes	Yes	Yes	Yes
SVC_FULFILORDREFDTL	Yes	Yes	Yes	Yes
SVC_FULFILORD	Yes	Yes	Yes	Yes
SVC_FULFILORDCUST	Yes	Yes	Yes	Yes
SVC_FULFILORDDTL	Yes	Yes	Yes	Yes
TSFHEAD	Yes	Yes	Yes	No
TSFDETAIL	Yes	Yes	Yes	No
ORDCUST	Yes	Yes	No	No
ORDCUST_DETAIL	Yes	Yes	Yes	No

TABLE	SELECT	INSERT	UPDATE	DELETE
ORDHEAD	Yes	Yes	Yes	Yes
ORDLOC	Yes	Yes	Yes	No
ORDSKU	Yes	Yes	No	No
ORDSKU_HTS	No	Yes	No	No
ORDSKU_HTS_ASSESS	No	Yes	No	No
ORDLOC_EXP	No	Yes	Yes	No
TSFHEAD	Yes	Yes	Yes	No
TSFDETAIL	Yes	Yes	Yes	No
TSFHEAD_L10N_EXT	No	Yes	Yes	No
ORDCUST_L10N_EXT	No	Yes	Yes	N o
ORDCUST_PUB_TEMP	No	Yes	No	No
ITEM_MASTER	Yes	No	No	No
ITEM_LOC_SOH	Yes	Yes	Yes	No
ITEM_LOC	Yes	No	No	No

Customer Order Item Substitution Service

Functional Area

Customer Orders

Business Overview

When a store is allowed to pick inventory to fulfill a Customer Order, when the inventory of the item ordered does not meet quality standards or is unavailable, then the order indicates that substitutions are allowed for that item. In that case, the store may choose to fulfill the order with a substitute item. If that occurs, SIM has the ability to substitute items on the Customer Order with another predefined Substitute Item.

In such cases, SIM notifies OMS through the SO Status message that an alternative item has been pushed into the order.

Based on the notification from SIM, OMS updates the customer order. OMS notifies RMS with the same details received from SIM so that RMS updates the inventory and customer order details. Based on OMS notification RMS updates the cancelled quantity for the original item and also creates the customer order reservation for the substitute item by updating the customer reserve inventory.

Package Impact

PL/SQL Web Service Wrapper

Package: CustOrdSubstituteServiceProvid

This layer is the entry point for calling the Customer Order Item Substitution webservice. The following operation is available:

```
createCustOrdSubColDesc (
                    I_serviceOperationContext IN OUT "RIB_ServiceOpContext_REC",
                    I_businessObject          IN  "RIB_CustOrdSubColDesc_REC",
                    O_serviceOperationStatus  OUT "RIB_ServiceOpStatus_REC",
                    O_businessObject          OUT "RIB_InvocationSuccess_REC"
                     )
```

- This procedure validates the input service operation context and initializes the output service operation status.
- Calls CREATE_CO_SUBSTITUTE to process the Customer Order Item Substitution message.
- Any failures (validation errors) encountered during the processing are passed back into the response object. If there are no failures, success status is returned.

Service Provider Layer

Package: SVCPROV_CUSTORDSUB

This layer, called from Web service wrapper, inserts the input business objects into the staging tables and calls the core business logic to process the request. The following operation is available.

```
CREATE_CO_SUBSTITUTE (O_serviceOperationStatus IN OUT "RIB_ServiceOpStatus_REC",
                    I_businessObject         IN     "RIB_CustOrdSubColDesc_REC")
```

- The count of detail message in the input business object is validated against the collection_size to make sure entire message has been received.
- The input business object is staged into the staging tables - SVC_CUSTORDSUB and SVC_CUSTORDSUBDTL.
- Calls the core business layer CREATE_CO_SUBSTITUTE to process the input item substitution request.
- In case of errors received from the core business logic, the error message from the staging table is retrieved and written to the failure table of the output business object.
- On successful processing, the processed data from the staging table is deleted.

Core Logic Layer

Package: CORESVC_CUSTORDSUB

The layer implements the core business logic for customer order subscription. The following operation is available.

```
CREATE_CO_SUBSTITUTE (O_error_message IN OUT  RTK_ERRORS.RTK_TEXT%TYPE,
                      I_process_id    IN      SVC_CUSTORDSUB.PROCESS_ID%TYPE,
                      I_chunk_id      IN      SVC_CUSTORDSUB.CHUNK_ID%TYPE)
```

- The data in the staging table is validated. The header table svc_custordsub is validated first and if there are no errors in the header data then the detail table svc_custordsubdtl gets validated. In case of errors, all the validation errors are written back to the staging table and the function returns back with error.
- Post successful validation, the customer order details are updated in ordcust_detail table. The cancelled quantity for the original ordered item is updated. New customer order detail record is created for the substituted item.
- The customer reserve bucket in item_loc_soh table is updated by making a call to CUSTOMER_RESERVE_SQL to release the reserved quantity for original item and increase the reserve quantity for the substituted item.
- The status in the staging table is updated to 'C'ompleted to indicate successful processing of the data.

Message XSD

Here are the filenames that correspond with each message type. Please consult the RIB documentation for each message type in order to get detailed information of the composition of each message.

Message Types	Message Type Description	XML Schema Definition (XSD)
CustOrdSubDesc	Customer Order Substitute Message	CustOrdSubDesc.xsd
CustOrdSubColDesc	Collection of Customer Order Substitute Message	CustOrdSubColDesc.xsd

Design Assumptions

- Substitution logic holds good only for the customer orders fulfilled from stores.
- Catchweight, Transformable, Consignment, Concession and Deposit container items are not supported for customer order item substitution.
- The quantities are always in Standard UOM.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
SVC_CUSTORDSUB	Yes	Yes	Yes	Yes
SVC_CUSTORDSUBDTL	Yes	Yes	Yes	Yes
ORDCUST	Yes	No	No	No
ORDCUST_DETAIL	Yes	Yes	Yes	No
ITEM_MASTER	Yes	No	No	No
DEPS	Yes	No	No	No
ITEM_LOC	Yes	Yes	No	No
ITEM_LOC_SOH	No	Yes	Yes	No
INV_RESV_UPDATE_TEMP	No	Yes	No	No

Inventory Detail Lookup Service

Functional Area

Inventory

Business Overview

This real-time inventory availability lookup facility provided by RMS can be used by external systems, such as an on-line order capture system (OOC) and order management system (OMS), to retrieve item/location inventory based on RMS's view of inventory at a point in time. RMS will provide this information for any warehouse or store which is valid for customer order sourcing/fulfillment via a Web service.

This Web service requires code to abstract the interface logic (service provider layer) from the business processing logic (core layer) and RMS packages will be used by the core layer to perform the actual validations and processing for inventory detail.

Package Impact

PL/SQL Web Service Wrapper

Package: InventoryDetailServiceProvider

This layer is the entry point for the inventory detail lookup Web service. The following operation is available:

```
lookupInvAvailCriVo(
             I_serviceOperationContext IN OUT "RIB_ServiceOpContext_REC",
              I_businessObject          IN     "RIB_InvAvailCriVo_REC",
              O_serviceOperationStatus  OUT    "RIB_ServiceOpStatus_REC",
              O_businessObject          OUT    "RIB_InvAvailColDesc_REC")
```

- This procedure validates the input service operation context and initializes the output service operation status.
- Calls GET_INV_DETAIL to get the inventory details for the input message.
- Any failures (validation errors) encountered during the processing are passed back into the response object. If there are no failures, success status is returned.

Service Provider Layer

Package: SVCPROV_INVAVAIL

This layer calls the core business layer to process the inventory lookup request. The following operation is available.

```
GET_INV_DETAIL(O_ServiceOperationStatus    IN OUT "RIB_ServiceOpStatus_REC",
               O_business_object           OUT    "RIB_InvAvailColDesc_REC",
               I_business_object           IN     "RIB_InvAvailCriVo_REC")
```

- Calls the core business layer CORESVC_INVAVAIL to process the inventory detail lookup request.
- In case of errors received from the core business logic, the error message is written to the failure table of the output business object.

Core Logic Layer

Package: CORESVC_INVAVAIL

This layer implements the core business logic for inventory detail lookup. The following operation is available.

```
GET_INV_DETAIL(O_error_message     OUT RTK_ERRORS.RTK_TEXT%TYPE,
               O_business_object OUT "RIB_InvAvailColDesc_REC",
               O_error_tbl       OUT SVCPROV_UTILITY.ERROR_TBL,
               I_business_object IN  "RIB_InvAvailCriVo_REC")
```

- The data in the input business object is validated. If validation errors are encountered, this layer returns the errors in a collection.
- The available inventory is fetched from RMS based on the following:
    - The available quantity is fetched from item_loc_soh as stock_on_hand – SUM of tsf_reserved_qty, customer_resv, rtv_qty and non_sellable_qty.
    - The warehouse inventory for physical warehouse/channel is only taken for customer orderable stockholding virtual warehouse under it.
    - If the inventory detail lookup is for a pack item at store, the pack inventory is estimated based on the maximum number of complete packs which can be created by using all the available inventory of its component. The pack_calculate_ind is set to 'Y' to indicate the pack inventory is estimated.

Message XSD

Below are the filenames that correspond with each message type. Please consult the RIB documentation for each message type in order to get a detailed picture of the composition of each message.

Message Types	Message Type Description	XML Schema Definition (XSD)
InvDetailCriVo	Inventory Detail Lookup Criteria	InvAvailCriVo.xsd
InvAvailDesc	Inventory Detail response Description	InvAvailDesc.xsd
InvAvailColDesc	Collection of Inventory Detail Description	InvAvailColDesc.xsd

Design Assumptions

- Catchweight, Transformable, Consignment, Concession and Deposit container items are not supported for available inventory lookup.
- This inventory detail lookup is only for customer orderable inventory.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_LOC_SOH	YES	NO	NO	NO
ITEM_MASTER	YES	NO	NO	NO
WH	YES	NO	NO	NO
STORE	YES	NO	NO	NO
CHANNELS	YES	NO	NO	NO
DEPS	YES	NO	NO	NO
PACKITEM_BREAKOUT	YES	NO	NO	NO

Inventory Back Order Service

Functional Area

Inventory

Business Overview

Retailers selling through ecommerce channels often take customer orders even if inventory is not available with the expectation of future inventory being available to fill the order. If an order is captured against future inventory the quantity is placed in 'Backordered' status in the external system and a backorder message is sent to RMS through the backorder Web service.

This Web service will update the backorder quantity in RMS. An external order management system will send backorder reserve requests to RMS when a customer fulfillment is made and backorder release requests when inventory is made available at the fulfillment location.

Package Impact

PL/SQL Web service Wrapper

Package: InventoryBackOrderServiceProvi

This layer is the entry point for calling the BackOrder. The following operation is available:

```
createInvBackOrdColDesc(
                    I_serviceOperationContext IN OUT "RIB_ServiceOpContext_REC",
                    I_businessObject          IN  "RIB_InvBackOrdColDesc_REC",
                    O_serviceOperationStatus  OUT "RIB_ServiceOpStatus_REC",
                    O_businessObject          OUT "RIB_InvocationSuccess_REC"
                     )
```

- This procedure validates the input service operation context and initializes the output service operation status.

- Calls CREATE_BACKORDER to process the Backorder message.
- Any failures (validation errors) encountered during the processing are passed back into the response object. If there are no failures, success status is returned.

Service Provider Layer

Package: SVCPROV_ INVBACKORD

This layer, called from Web service wrapper, inserts the input business objects into the staging tables and calls the core business logic to process the request. The following operation is available.

```
CREATE_BACKORDER (O_serviceOperationStatus IN OUT "RIB_ServiceOpStatus_REC",
                  I_businessObject         IN     "RIB_InvBackOrdColDesc_REC")
```

The count of detail records in the input business object is validated against the collection_size to make sure entire message has been received.

- The input business object is staged into the staging table - SVC_INVBACKORD.
- Calls the core business layer CREATE_BACKORDER to process the input backorder request.
- In case of errors received from the core business logic, the error message from the staging table is retrieved and written to the failure table of the output business object.
- On successful processing, the processed data from the staging table is deleted.

Core Logic Layer

Package: CORESVC_ INVBACKORD

The layer implements the core business logic for backorder subscription. The following operation is available.

```
CREATE_BACKORDER (O_error_message IN OUT  RTK_ERRORS.RTK_TEXT%TYPE,
                  I_process_id    IN      SVC_CUSTORDSUB.PROCESS_ID%TYPE,
                  I_chunk_id      IN      SVC_CUSTORDSUB.CHUNK_ID%TYPE)
```

- The data in the staging table svc_invbackord is validated. In case of errors, all the validation errors are written back to the staging table and the function returns back with error.
- Post successful validation, the backorder details are updated in the item_loc_soh table. The backorder details are updated for both Stores and Warehouses. In case of warehouses, the corresponding virtual warehouses are fetched and the details are updated accordingly.
- The customer backorder bucket or pack comp customer backorder bucket in the item_loc_soh table are updated based on the input request item being a regular item or pack item.
- In the case of a negative quantity (which indicates backorder release) in the input message, the quantity is subtracted from the customer backorder column in item_loc_soh indicating the release of backorder when the quantity is available in the fulfillment location. In this case also, for warehouses, the corresponding virtual warehouses will be identified from which the quantity has to be released.

- An insert is made into inv_resv_update_temp table for a location which has been backordered for the current day. This table is used by inventory extract to AIP to identify the location for which the inventory feed to AIP should be extracted signifying a change in inventory or back order position.
- The status in the staging table is updated to 'C'ompleted to indicate successful processing of the data.

Message XSD

Below are the filenames that correspond with each message type. Please consult the RIB documentation for each message type in order to get a detailed picture of the composition of each message.

Message Types	Message Type Description	XML Schema Definition (XSD)
InvBackOrdDesc	Inventory Backorder Message	InvBackOrdDesc.xsd
InvBackOrdColDesc	Collection of Inventory Backorder Message	InvBackOrdColDesc.xsd

Design Assumptions

- Catchweight, Transformable, Consignment, Concession and Deposit container items are not supported for backorder requests.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
SVC_INVBACKORD	Yes	Yes	Yes	Yes
INV_RESV_UPDATE_TEMP	Yes	Yes	No	No
STORE	Yes	No	No	No
WH	Yes	No	No	No
CHANNELS	Yes	No	No	No
DEPS	Yes	No	No	No
UOM_CLASS	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
ITEM_LOC_SOH	Yes	No	Yes	No
V_PACKSKU_QTY	Yes	No	No	No

Pricing Cost Lookup Service

Functional Area

Foundation Data

Business Overview

This Web service used by RMS to expose pricing cost information to external systems. The primary user of this information is assumed to be an Order Management System (OMS), which manages wholesale customer orders and needs visibility to cost information as part of the negotiation process for margin visibility. This API supports providing cost information for an item/location or item/supplier/location.

Package Impact

PL/SQL Web Service Wrapper

Package: PricingCostServiceProviderImpl

This layer is the entry point for calling the Pricing Cost Web service. The following available operation is:

```
lookupPrcCostColCriVo(I_serviceOperationContext IN OUT "RIB_ServiceOpContext_REC",
                      I_businessObject          IN  "RIB_PrcCostColCriVo_REC",
                      O_serviceOperationStatus  OUT "RIB_ServiceOpStatus_REC",
                      O_businessObject          OUT "RIB_PrcCostColDesc_REC")
```

This procedure:

- Validates the input service operation context and initializes the output service operation status
- Calls SVCPROV_PRICECOST_SQL.GET_PRICING_COST to get the pricing cost

Any failures (validation errors) encountered during the processing are passed back into the response object. If there are no failures, success status is returned.

Service Provider Layer

Package: SVCPROV_ PRICECOST_SQL

This layer is called from Web service wrapper which calls the core business logic to process the request.

```
CORESVC_PRICECOST_SQL.GET_PRICING_COST
            (O_error_message     OUT   RTK_ERRORS.RTK_TEXT%TYPE,
             O_business_object   OUT   "RIB_PrcCostColDesc_REC",
             O_error_tbl         OUT   SVCPROV_UTILITY.ERROR_TBL,
             I_business_object   IN    "RIB_PrcCostColCriVo_REC")
```

- In case of errors received from the core business logic, the error message from the staging table is retrieved and written to the failure table of the output business object.
- On successful processing, the output object `O_business_object` will have the pricing cost.

Core Logic Layer

Package: CORESVC_PRICECOST_SQL

This layer implements the core business logic to get the pricing cost. The following operation is available.

```
GET_PRICING_COST(O_error_message     OUT   RTK_ERRORS.RTK_TEXT%TYPE,
                 O_business_object   OUT   "RIB_PrcCostColDesc_REC",
                 O_error_tbl         OUT   SVCPROV_UTILITY.ERROR_TBL,
                 I_business_object   IN    "RIB_PrcCostColCriVo_REC")
```

- The data in the input business object is validated. In case of errors, all the validation errors are written back to the staging table and the function returns back with error.
- Post successful validation, the pricing cost field is returned from the FUTURE_COST table with all the input fields as output business object.

Message XSD

This table contains the filenames that correspond with each message type. Please consult the RIB documentation for each message type in order to get a detailed picture of the composition of each message.

Message Types	Message Type Description	XML Schema Definition (XSD)
PrcCostCriVo	Collection of input parameters	PrcCostCriVo.xsd
PrcCostDesc	Collection of input parameters with Pricing_cost	PrcCostDesc.xsd

Design Assumptions

- Only Approved and transaction level items are valid.
- Location must be company store or physical warehouse that is customer orderable.
- Since FUTURE_COST only holds virtual warehouses, the filter criteria to get the correct virtual warehouse will be:
- Channel ID match,
- Low number Channel type match not protected.
- Low number Channel type match protected.
- Primary not protected.
- Primary protected.
- If there are multiple results for any level match, the lowest number virtual warehouse in the result set will be the one returned. In case a match is not found, the next warehouse returned will be passed till a match is found. If no match found the record gets rejected.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_MASTER	Yes	No	No	No
COUNTRY	Yes	No	No	No
SUPS	Yes	No	No	No
STORE	Yes	No	No	No
CHANNELS	Yes	No	No	No
WH	Yes	No	No	No
FUTURE_COST	Yes	No	No	No

Customer Credit Check Web Service

Functional Area

Franchise

Business Overview

RMS provides an interface to update the credit_ind for a franchise customer in RMS. A credit check for the franchisee will be performed before a franchisee order can be approved. Credit_ind is a column in WF_CUSTOMER table. It determines whether customer has a good credit. Valid values are 'Y' and 'N'. For each customer_id/customer_group_id combination, the credit_ind should be updated in the WF_CUSTOMER table. RMS supports Web service for processing Customer Credit Check message from external financial application. RMS exposes a 'CustCreditcheckservice' Web service to update the credit_ind. An external financial application will invoke the service with input collection to update the credit_ind in RMS table.

Web Service Deployment

- Accepts a collection of input parameters. The input is a collection of customer_id, customer_group_id and credit_ind. For each customer_id/customer_group_id combination, the credit_ind should be updated in the WF_CUSTOMER table.
- RMS returns failure status as part of the response object in the Web service call if credit_ind is not updated due to validation errors.

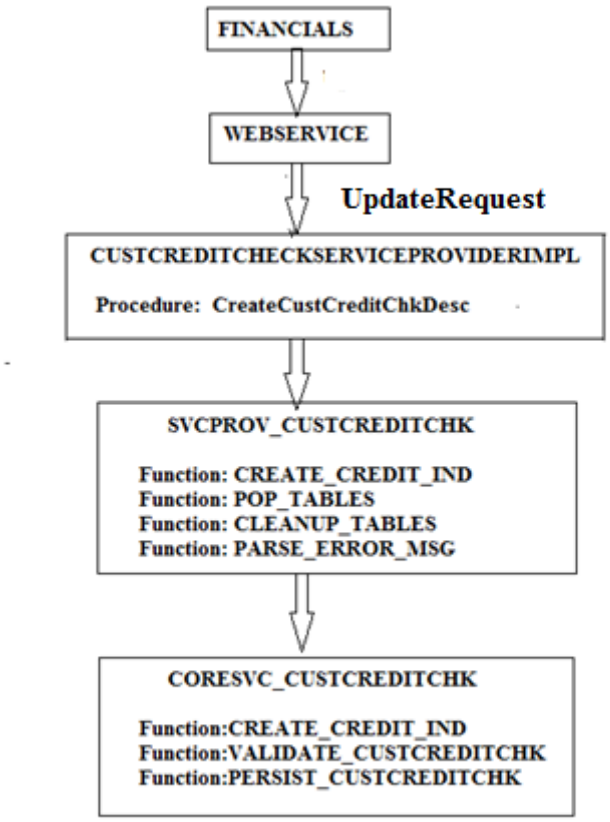
Package Impact

Filename: CustCreditCheckServiceProviderImplSpec and

CustCreditCheckServiceproviderImplBody

The Web service 'CustomerCreditCheckService' with an operation of 'updateCustCredit' is available for external financial application to send update request for credit_ind to RMS. The Web service invokes the public interface 'CreateCustCreditChkColDesc' (CustCreditCheckServiceProviderImpl.CreateCustCreditChkColDesc).
The input is a collection of customer_id, customer_group_id and credit_ind. For each customer_id/customer_group_id combination, the credit_ind should be updated in the WF_CUSTOMER table. This public interface will call svcprov_context.set_svcprov_context and svcprov_custcreditchk.create_credit_ind to update the credit_ind in WF_CUSTOMER table. The messages are staged, validated and persisted to RMS using the core functions. At the end of the processing, the staging tables are purged and a confirmation status is returned.

The flowchart below illustrates the complete process:



Process Flow for Customer Credit Check Web Service

Design Assumptions

- Record should be present in WF_CUSTOMER table for the given wf_customer_id and wf_customer_group_id.
- Credit_ind will not be updated if there is any validation error. Only Approved and transaction level items are valid.

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
SVC_CUSTCREDITCHK	Yes	Yes	No	Yes
WF_CUSTOMER	Yes	No	Yes	No
WF_CUSTOMER_GROUP	Yes	No	No	No

Store Order Subscription API

Functional Area

Procurement

Business Overview

The Web service provided by RMS is used by external systems, such as SIM to perform the following:

- Create, Modify, and Delete a Store Order (PO or Transfer) details in RMS.
- Retrieve Item Sales information for Store and Item.
- Retrieve Deals information for Store and Item.
- Retrieve Store Orders (PO or Transfers) for the Store.
- Retrieve details of Store Order (PO or Transfer).

The Store Order Web service replaces the service interface based on RSL .This Web service requires code to abstract the interface logic (service provider layer) from the business processing logic (core layer), which uses RMS packages to perform the processing for a requested operations.

Package Impact

PL/SQL Web Service Wrapper

Package: StoreOrderServiceProviderImpl

This layer is the entry point for the Store Order Web service. The following available operations are:

1. createLocPOTsfDesc(
```
I_serviceOperationContext IN OUT "RIB_ServiceOpContext_REC",
I_businessObject          IN  "RIB_LocPOTsfDesc_REC",
O_serviceOperationStatus  OUT "RIB_ServiceOpStatus_REC",
O_businessObject          OUT "RIB_LocPOTsfRef_REC")
```
   - This procedure validates the input service operation context and initializes the output service operation status.
   - Calls SVCPROV_STOREORDER.CREATE_LOC_PO_TSF for creating the Store Order (PO or Transfer) details.
   - The generated order number is returned in the O_businessObject.
   - Any failures (validation errors) encountered during the processing are passed back into the ServiceOpStatus response object. If there are no failures, success status is returned through ServiceOpStatus.

2. deleteLocPOTsfDesc(
```
I_serviceOperationContext  IN OUT  "RIB_ServiceOpContext_REC",
I_businessObject           IN     "RIB_LocPOTsfDesc_REC",
O_serviceOperationStatus   OUT    "RIB_ServiceOpStatus_REC",
O_businessObject           OUT "RIB_InvocationSuccess_REC")
```
   - This procedure validates the input service operation context and initializes the output service operation status.
   - Calls SVCPROV_STOREORDER.DELETE_LOC_PO_TSF for deleting the Store Order (PO or Transfer) details.
   - Any failures (validation errors) encountered during the processing are passed back to the ServiceOpStatus response object. If there are no failures, success status is returned through ServiceOpStatus.

3. modifyLocPOTsfDesc(
```
I_serviceOperationContext  IN OUT "RIB_ServiceOpContext_REC",
I_businessObject           IN     "RIB_LocPOTsfDesc_REC",
O_serviceOperationStatus   OUT    "RIB_ServiceOpStatus_REC",
O_businessObject           OUT    "RIB_InvocationSuccess_REC")
```
   - This procedure validates the input service operation context and initializes the output service operation status.
   - Calls SVCPROV_STOREORDER.MOD_LOC_PO_TSF for modifying of the Store Order (PO or Transfer) details.
   - Any failures (validation errors) encountered during the processing are passed back into the ServiceOpStatus response object. If there are no failures, success status is returned through ServiceOpStatus.

4. createDetailLocPOTsfDesc(

```
I_serviceOperationContext  IN OUT "RIB_ServiceOpContext_REC",
I_businessObject           IN  "RIB_LocPOTsfDesc_REC",
O_serviceOperationStatus   OUT "RIB_ServiceOpStatus_REC",
O_businessObject           OUT "RIB_InvocationSuccess_REC")
```

   - This procedure validates the input service operation context and initializes the output service operation status.
   - Calls SVCPROV_STOREORDER.CREATE_LOC_PO_TSF_DETAIL for creating the Store Order (PO or Transfer) details.
   - Any failures (validation errors) encountered during the processing are passed back into the ServiceOpStatus response object. If there are no failures, success status is returned through ServiceOpStatus.

5. deleteDetailLocPOTsfDesc(

```
I_serviceOperationContext  IN OUT  "RIB_ServiceOpContext_REC",
I_businessObject           IN      "RIB_LocPOTsfDesc_REC",
O_serviceOperationStatus   OUT     "RIB_ServiceOpStatus_REC",
O_businessObject           OUT     "RIB_InvocationSuccess_REC")
```

   - This procedure validates the input service operation context and initializes the output service operation status.
   - Calls SVCPROV_STOREORDER. DELETE_LOC_PO_TSF_DETAIL for deleting of Store Order (PO or Transfer) details.
   - Any failures (validation errors) encountered during the processing are passed back into the ServiceOpStatus response object. If there are no failures, success status is returned through ServiceOpStatus.

6. modifyDetailLocPOTsfDesc(

```
I_serviceOperationContext  IN OUT "RIB_ServiceOpContext_REC",
I_businessObject           IN     "RIB_LocPOTsfDesc_REC",
O_serviceOperationStatus   OUT    "RIB_ServiceOpStatus_REC",
O_businessObject           OUT    "RIB_InvocationSuccess_REC")
```

   - This procedure validates the input service operation context and initializes the output service operation status.
   - Calls SVCPROV_STOREORDER.MOD_LOC_PO_TSF_DETAIL for modification of Store Order (PO or Transfer) details.
   - Any failures (validation errors) encountered during the processing are passed back into the ServiceOpStatus response object. If there are no failures, success status is returned through ServiceOpStatus.

7. queryDeal(

```
I_serviceOperationContext  IN OUT  "RIB_ServiceOpContext_REC",
I_businessObject           IN      "RIB_LocPOTsfDealsCriVo_REC",
O_serviceOperationStatus   OUT     "RIB_ServiceOpStatus_REC",
O_businessObject           OUT     "RIB_LocPOTsfDealsColDesc_REC")
```

   - This procedure validates the input service operation context and initializes the output service operation status
   - Calls SVCPROV_STOREORDER.QUERY_LOC_PO_TSF_DEALS for retrieving the deals information which is returned to the O_businessObject.
   - Any failures (validation errors) encountered during the processing are passed back into the ServiceOpStatus response object. If there are no failures, success status is returned through ServiceOpStatus.

8. queryItemSales(
```
I_serviceOperationContext  IN OUT  "RIB_ServiceOpContext_REC",
I_businessObject           IN      "RIB_LocPOTsfItmSlsCriVo_REC",
O_serviceOperationStatus   OUT     "RIB_ServiceOpStatus_REC",
O_businessObject           OUT     "RIB_LocPOTsfItmSlsColDesc_REC")
```
   - This procedure validates the input service operation context and initializes the output service operation status.
   - Calls SVCPROV_STOREORDER.QUERY_LOC_PO_TSF_ITEMSALES for retrieving the item sales information which is returned to the O_businessObject.
   - Any failures (validation errors) encountered during the processing are passed back into the ServiceOpStatus response object. If there are no failures, success status is returned through ServiceOpStatus.

9. queryStoreOrder (
```
I_serviceOperationContext  IN OUT "RIB_ServiceOpContext_REC",
I_businessObject           IN     "RIB_LocPOTsfHdrCriVo_REC",
O_serviceOperationStatus   OUT    "RIB_ServiceOpStatus_REC",
O_businessObject           OUT    "RIB_LocPOTsfHdrColDesc_REC")
```
   - This procedure validates the input service operation context and initializes the output service operation status.
   - Calls SVCPROV_STOREORDER.QUERY_LOC_PO_TSF_HEADER for retrieving the store orders(PO, Transfer) .This will be returned in the O_businessObject.
   - Any failures (validation errors) encountered during the processing are passed back into the ServiceOpStatus response object. If there are no failures, success status is returned through ServiceOpStatus.

10. queryStoreOrderDetail (
```
I_serviceOperationContext  IN OUT  "RIB_ServiceOpContext_REC",
I_businessObject           IN      "RIB_LocPOTsfDtlsCriVo_REC",
O_serviceOperationStatus   OUT     "RIB_ServiceOpStatus_REC",
O_businessObject           OUT     "RIB_LocPOTsfDesc_REC")
```
   - This procedure validates the input service operation context and initializes the output service operation status.
   - Calls SVCPROV_STOREORDER.QUERY_LOC_PO_TSF_DETAIL for retrieving the details of the store orders (PO, Transfer) .This will be returned in the O_businessObject.
   - Any failures (validation errors) encountered during the processing are passed back into the ServiceOpStatus response object. If there are no failures, success status is returned through ServiceOpStatus.

Service Provider Layer

Package: SVCPROV_STOREORDER

This layer calls the core business layer to process the requests. The following operations are available.

1. CREATE_LOC_PO_TSF(
```
O_serviceOperationStatus    IN OUT "RIB_ServiceOpStatus_REC",
O_businessObject            OUT    "RIB_LocPOTsfRef_REC",
I_businessObject            IN     "RIB_LocPOTsfDesc_REC")
```
   - Calls the function CREATE_MOD_LOCPO in the core business layer CORESVC_STOREORDER to process the Store Order creation request.

- In case of errors received from the core business logic, the error message is written to the failure table of the O_serviceOperationStatus object.

2. CREATE_LOC_PO_TSF_DETAIL(
   O_serviceOperationStatus IN OUT "RIB_ServiceOpStatus_REC",
   I_businessObject IN "RIB_LocPOTsfDesc_REC")

   - Calls the function CREATE_MOD_LOCPO in the core business layer CORESVC_STOREORDER to process the Store Order Details creation request.
   - In case of errors received from the core business logic, the error message is written to the failure table of the O_serviceOperationStatus object.

3. MOD_LOC_PO_TSF(
   O_serviceOperationStatus IN OUT "RIB_ServiceOpStatus_REC",
   I_businessObject IN "RIB_LocPOTsfDesc_REC")

   - Calls the function CREATE_MOD_LOCPO in the core business layer CORESVC_STOREORDER to process the Store Order Modification request.
   - In case of errors received from the core business logic, the error message is written to the failure table of the O_serviceOperationStatus object.

4. MOD_LOC_PO_TSF_DETAIL(
   O_serviceOperationStatus IN OUT "RIB_ServiceOpStatus_REC",
   I_businessObject IN "RIB_LocPOTsfDesc_REC")

   - Calls the function CREATE_MOD_LOCPO in the core business layer CORESVC_STOREORDER to process the Store Order details modification request.
   - In case of errors received from the core business logic, the error message is written to the failure table of the O_serviceOperationStatus object.

5. DELETE_LOC_PO_TSF(
   O_serviceOperationStatus IN OUT "RIB_ServiceOpStatus_REC",
   I_businessObject IN "RIB_LocPOTsfDesc_REC")

   - Calls the function DEL_LOCPO in the core business layer CORESVC_STOREORDER to process the Store Order deletion request.
   - In case of errors received from the core business logic, the error message is written to the failure table of the O_serviceOperationStatus object.

6. DELETE_LOC_PO_TSF_DETAIL(
   O_serviceOperationStatus IN OUT "RIB_ServiceOpStatus_REC",
   I_businessObject IN "RIB_LocPOTsfDesc_REC")

   - Calls the function DEL_LOCPO in the core business layer CORESVC_STOREORDER to process the Store Order Details deletion request.
   - In case of errors received from the core business logic, the error message is written to the failure table of the O_serviceOperationStatus object.

7. QUERY_LOC_PO_TSF_DEALS(
   O_serviceOperationStatus IN OUT "RIB_ServiceOpStatus_REC",
   O_businessObject OUT "RIB_LocPOTsfDealsColDesc_REC",
   I_businessObject IN "RIB_LocPOTsfDealsCriVo_REC")

   - Calls the function GET_DEALS in the core business layer CORESVC_STOREORDER to process the Deals query request.
   - In case of errors received from the core business logic, the error message is written to the failure table of the O_serviceOperationStatus object.

8. QUERY_LOC_PO_TSF_ITEMSALES(
   O_serviceOperationStatus IN OUT "RIB_ServiceOpStatus_REC",
   O_businessObject OUT "RIB_LocPOTsfItmSlsColDesc_REC",
   I_businessObject IN "RIB_LocPOTsfItmSlsCriVo_REC")
   - Calls the function GET_ITEMS_SALES in the core business layer CORESVC_STOREORDER to process the Item Sales retrieval request.
   - In case of errors received from the core business logic, the error message is written to the failure table of the O_serviceOperationStatus object.
9. QUERY_LOC_PO_TSF_HEADER (
   O_serviceOperationStatus IN OUT "RIB_ServiceOpStatus_REC",
   O_businessObject OUT "RIB_LocPOTsfRef_REC",
   I_businessObject IN "RIB_LocPOTsfDesc_REC")
   - Calls the function GET_STORE_ORDERS in the core business layer CORESVC_STOREORDER to process the Store Orders retrieval request.
   - In case of errors received from the core business logic, the error message is written to the failure table of the O_serviceOperationStatus object.
10. QUERY_LOC_PO_TSF_DETAIL(
    O_serviceOperationStatus IN OUT "RIB_ServiceOpStatus_REC",
    O_businessObject OUT "RIB_LocPOTsfDesc_REC",
    I_businessObject IN "RIB_LocPOTsfDtlsCriVo_REC")
    - Calls the function GET_STORE_ORDER_DETAILS in the core business layer CORESVC_STOREORDER to process the Store Order Details retrieval request.
    - In case of errors received from the core business logic, the error message is written to the failure table of the O_serviceOperationStatus object.

Core Logic Layer

Package: CORESVC_STOREORDER

This layer implements the core business logic for Store Order operations. The following operations are available:

1. CREATE_MOD_LOCPO(
   IO_error_message IN OUT RTK_ERRORS.RTK_TEXT%TYPE,
   O_locpotsfref_rec OUT NOCOPY "RIB_LocPOTsfRef_REC",
   I_locpodesc_rec IN "RIB_LocPOTsfDesc_REC",
   I_action IN VARCHAR2)
   RETURN BOOLEAN
   - This function is used to Create, Modify Transfer/ PO header and details based on the input action_type. It performs input payload validation and calls RMSSUB_XORDER or RMSSUB_XTSF APIs to achieve it.
   - The encountered error will be returned on IO_error_message.
   - For a create operation, the generated order_no or tsf_no will be returned as the order_id in the LocPOTsfRef object.

2. DEL_LOCPO(
   IO_error_message IN OUT RTK_ERRORS.RTK_TEXT%TYPE,
   I_locpotsfdesc_rec IN "RIB_LocPOTsfDesc_REC",
   I_action IN VARCHAR2)
   RETURN BOOLEAN
   - The function is used to delete transfer/PO header and details based on the input action_type. It performs input payload validation and calls RMSSUB_XORDER or RMSSUB_XTSF APIs to achieve it.
   - The encountered error will be returned on O_error_message.
3. GET_DEALS(
   IO_error_message IN OUT RTK_ERRORS.RTK_TEXT%TYPE,
   O_locpotsfdealscoldesc_rec OUT NOCOPY "RIB_LocPOTsfDealsColDesc_REC",
   I_locpotsfdealscrivo_rec IN "RIB_LocPOTsfDealsCriVo_REC")
   RETURN BOOLEAN
   - This function retrieves all applicable deals information for the input item/store.
   - The encountered error will be returned on O_error_message.
4. GET_ITEMS_SALES(
   IO_error_message IN OUT RTK_ERRORS.RTK_TEXT%TYPE,
   O_locpotsfitmslscoldesc_rec OUT NOCOPY RIB_LocPOTsfItmSlsColDesc_REC",
   I_locpoitmslsreq_rec IN "RIB_LocPOTsfItmSlsCriVo_REC")
   RETURN BOOLEAN
   - This function retrieves the item sales information for the input item/store.
   - The encountered error is returned on O_error_message.
5. GET_STORE_ORDERS(
   IO_error_message IN OUT RTK_ERRORS.RTK_TEXT%TYPE,
   O_locpotsfhdrcoldesc_rec OUT NOCOPY "RIB_LocPOTsfHdrColDesc_REC",
   I_locpotsfhdrcrivo_rec IN "RIB_LocPOTsfHdrCriVo_REC")
   RETURN BOOLEAN
   - The function retrieves all relevant store orders (POs and transfers) for the input store/item contained in the input business object.
   - The encountered error is returned on O_error_message.
6. GET_STORE_ORDER_DETAILS(
   IO_error_message IN OUT RTK_ERRORS.RTK_TEXT%TYPE,
   O_locpotsfdesc_rec OUT NOCOPY "RIB_LocPOTsfDesc_REC",
   I_locpotsfdtlscrivo_rec IN "RIB_LocPOTsfDtlsCriVo_REC")
   RETURN BOOLEAN
   - This is used to retrieve the store order details for the input store, order_no or tsf_no contained in the input business object.
   - The encountered error is returned on O_error_message.

Message XSD

Below are the filenames that correspond with each message type. Please consult the RIB documentation for each message type in order to get a detailed picture of the composition of each message.

Message Types	Message Type Description	XML Schema Definition (XSD)
LocPOTsfDesc	Location PO Transfer Description	LocPOTsfDesc.xsd
LocPOTsfRef	Location PO Transfer Reference	LocPOTsfRef.xsd
LocPOTsfHdrCriVo	Location PO Transfer Header query Criteria	LocPOTsfHdrCriVo.xsd
LocPOTsfHdrDesc	Location PO Transfer Header Description	LocPOTsfHdrDesc.xsd
LocPOTsfHdrColDesc	Collection of Location PO Transfer Header Description	LocPOTsfHdrColDesc.xsd
LocPOTsfDtlsCriVo	Location PO Transfer Details query criteria	LocPOTsfDtlsCriVo.xsd
LocPOTsfDealsCriVo	Location PO Transfer Deals query Criteria	LocPOTsfDealsCriVo.xsd
LocPOTsfDealsDesc	Location PO Transfer Deals Description	LocPOTsfDealsDesc.xsd
LocPOTsfDealsColDesc	Collection of Location PO Transfer Deals Description	LocPOTsfDealsColDesc.xsd
LocPOTsfItmSlsCriVo	Location PO Transfer Item Sales query Criteria	LocPOTsfItmSlsCriVo.xsd
LocPOTsfItmSlsDesc	Location PO Transfer Item Sales Description	LocPOTsfItmSlsDesc.xsd
LocPOTsfItmSlsColDesc	Collection of Location PO Transfer Item Sales Description	LocPOTsfItmSlsColDesc.xsd

Design Assumptions

- To minimize the development and testing effort, no staging tables have been used to stage the data.
- Instead of following the pattern of VALIDATE/ POPULATE/ PERSIST, existing package logic has been reused in the core business layer.
- Instead of returning with all the validation errors found, package will just return back the first error encountered.

Tables

TABLE	SELECT	INSERT	UPDATE	DELETE
ORDHEAD	YES	YES	YES	NO
ORDLOC	YES	YES	YES	YES
ORDSKU	YES	YES	YES	YES
TSFHEAD	YES	YES	YES	NO
TSFDETAIL	YES	YES	YES	YES
ITEM_LOC_HIST	YES	NO	NO	NO
DEAL_HEAD	YES	NO	NO	NO
DEAL_DETAIL	YES	NO	NO	NO
DEAL_THRESHOLD	YES	NO	NO	NO
ITEM_MASTER	YES	NO	NO	NO
STORE	YES	NO	NO	NO
WH	YES	NO	NO	NO

Item Reservation Service API

Functional Area

Item

Design Overview

The Item Reservation web service allows the external systems such as Assortment Planning or Category Management to reserve item numbers in RMS.

This web service contains the following:

- Item number type
- Quantity
- Days until expiry

The web service validates the item number type passed is a valid number type in RMS that supports auto generation of item numbers and the days until expiry is greater than zero.

Once validated, RMS should auto generate the requested quantity of item numbers of the specified type and store them in a reservation log table that tracks their usage and expiry. The columns in the reservation log table includes:

- Item number type
- Item number
- Expiry date = vdate when request is received+ Days until expiry

The following are the only valid item number types:

- ITEM
- UPC-A
- UPC-AS
- EAN13

The details inserted into the reservation log for every valid web service call would also be sent back as part of the web service response to inform the calling application that the Reservation was successful and the numbers are reserved. These pre-reserved numbers should not be considered for any auto-generated item numbers in RMS. If a user keys in a pre-reserved item number while creating a new item, the system should stop from proceeding by displaying an appropriate message indicating that this number is already reserved.

For more information, see *Oracle Retail Merchandising System Operations Guide, Volume 1 - Batch Overviews and Designs*.

The structure of the Item Reservation message family are as follows:

"RIB_ItemNumCriVo_REC" (input)
-- item_number_type
-- quantity
-- expiry_days
"RIB_ItemNumColDesc_REC" (output)
-- collection size
-- ItemNumDesc_TBL
"RIB_ItemNumDesc_REC"
-- item
-- item_number_type

Package Impact

API Modules

Filename: ItemManagementServiceProviderImplSpec/Body.sql

```
PROCEDURE reserveItemNumber(I_serviceOperationContext IN OUT
"RIB_ServiceOpContext_REC",
                  I_businessObject          IN  "RIB_ItemNumCriVo_REC",
                  O_serviceOperationStatus  OUT "RIB_ServiceOpStatus_REC",
                  O_businessObject          OUT "RIB_ItemNumColDesc_REC")
```

This will call the Item Reservation Package - CORESVC_ITEM_RESERVE_SQL.RESERVE_ITEM_NUMBER().

Filename: Coresvc_itemreserveb/s.pls

```
RESERVE_ITEM_NUMBER(O_error_message  IN OUT     RTK_ERRORS.RTK_TEXT%TYPE,
                  O_businessObject IN OUT     "RIB_ItemNumColDesc_REC",
                  I_serviceOperationContext IN "RIB_ServiceOpContext_REC",
                  I_businessObject          IN "RIB_ItemNumCriVo_REC")
```

This package will be called by the web service to accept reservation requests that contain the following details:

- Item number type
- Quantity
- Days until expiry

The VALIDATE_FIELDS() checks if all the required fields are entered, calls the CHECK_ITEM_NUMBER_TYPE() and validate if the days until expiry is greater than zero. This function will validate whether the item number passed is a valid number type that supports auto generation of item numbers. Any invalid record displays an error.

If the record is valid, item numbers are generated depending on the item number type passed by calling the ITEM_NUMBER_TYPE_SQL.GET_NEXT() function. This call is be looped depending on the quantity.

The POPULATE_ITEM_RESERVE function inserts any valid records into the reservation log table (SVC_ITEM_RESERVATION).

Message XSD

Here are the filenames that correspond with each message type. Refer the RIB documentation for each message type for details of the composition of each message.

Message Types	Message Type Description	XML Schema Definition (XSD)
reserveItemNumber	Item Number Reservation	ItemNumCriVo.xsd

Design Assumptions

NA

Tables

TABLE	SELECT	INSERT	UPDATE	DELETE
SVC_ITEM_RESERVATION	Yes	Yes	No	Yes

5

Web Service Consumer Implementation

This chapter gives an overview about the Web service Consumer Implemenation API designs used in the RMS environment and various functional attributes used in the APIs.

GL Account Validation Service

Functional Area

Financial Integration

Business Overview

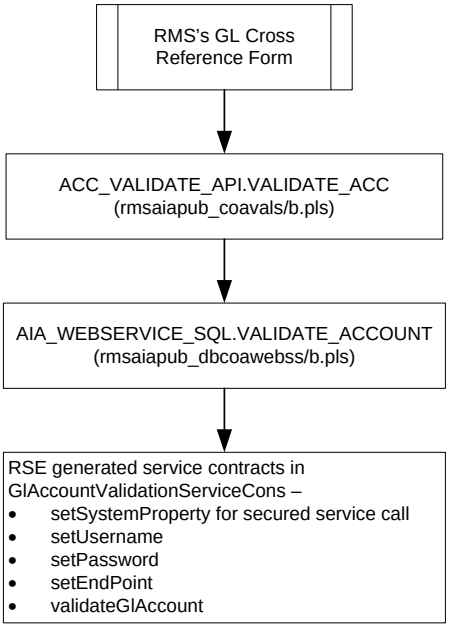
RMS holds the general ledger chart of account (GLCOA) information in the FIF_GL_ACCT table. A chart of account is essentially the financial application's debit and credit account segments (for example, company, cost center, account, and so on) that apply to the RMS product hierarchy. In some financial applications, this is known as code combination IDs (CCID). The GL Cross Reference form is then used to associate the appropriate department, class, subclass, and location data to a CCID and to populate that data to the GL_FIF_CROSS_REF table.

From RMS's GL Cross Reference form, RMS invokes a GL Account Validation Web service to validate the general ledger chart of accounts information against an external financial application. The segments like department, class, subclass and location cross reference to a CCID can only be established if the account is valid for the same segment combination in financial application.

This document describes the GL Account Validation Web service integration between RMS and an external financial application. In this integration context, RMS acts as the service consumer that invokes a Web service hosted by an external financial application.

Package Impact

The process flow for the Web service API.



Web Services API

Public API Procedures

Filename: rmsaiapub_coavals/b.pls

```
ACC_VALIDATE_API.VALIDATE_ACC
    (O_error_message    OUT    RTK_ERRORS.RTK_TEXT%TYPE,
     O_acc_val_rec   IN OUT    ACC_VALIDATE_API.ACC_VALIDATE_TBL)
RETURN BOOLEAN;
```

This function validates the segments, set_of_book_id and ccid combination in the input collection. It invokes AIA_WEBSERVICE_SQL.VALIDATE_ACCOUNT to do that.

Filename: rmsaiapub_dbcoawebss/b.pls

```
VALIDATE_ACCOUNT
    (O_error_message        IN OUT   RTK_ERRORS.RTK_TEXT%TYPE,
     O_requesting_system    IN OUT   VARCHAR2,
     O_set_of_books_id      IN OUT   ORG_UNIT.SET_OF_BOOKS_ID%TYPE,
     O_ccid                 IN OUT   FIF_GL_CROSS_REF.DR_CCID%TYPE,
     O_segment1             IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
     O_segment2             IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
     O_segment3             IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
     O_segment4             IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
     O_segment5             IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
     O_segment6             IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
     O_segment7             IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
     O_segment8             IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
     O_segment9             IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
     O_segment10            IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
     O_segment11            IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
     O_segment12            IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
     O_segment13            IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
     O_segment14            IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
     O_segment15            IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
     O_segment16            IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
```

```
    O_segment17           IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    O_segment18           IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    O_segment19           IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    O_segment20           IN OUT   FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    O_account_status      IN OUT   VARCHAR2,
    I_requesting_system   IN       VARCHAR2,
    I_set_of_books_id     IN       ORG_UNIT.SET_OF_BOOKS_ID%TYPE,
    I_ccid                IN       FIF_GL_CROSS_REF.DR_CCID%TYPE,
    I_segment1            IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment2            IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment3            IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment4            IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment5            IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment6            IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment7            IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment8            IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment9            IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment10           IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment11           IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment12           IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment13           IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment14           IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment15           IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment16           IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment17           IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment18           IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment19           IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE,
    I_segment20           IN       FIF_GL_CROSS_REF.DR_SEQUENCE1%TYPE)
RETURN BOOLEAN;
```

This function validates the segments, set_of_book_id and ccid combination in the input. It invokes the RSE generated service contract as defined in the GlAccountValidationServiceCons package to make a secure Web service call.

The service function that validates the account (validateGlAccount takes an OBJ_GLACCTCOLDESC as the input and returns OBJ_GLACCTCOLREF as the output. The structure of the objects is also defined in the service contract.

Message XSD

NA

Design Assumptions

NA

Table Impact

NA

6

ReSTful Web Service Implementation for RMS

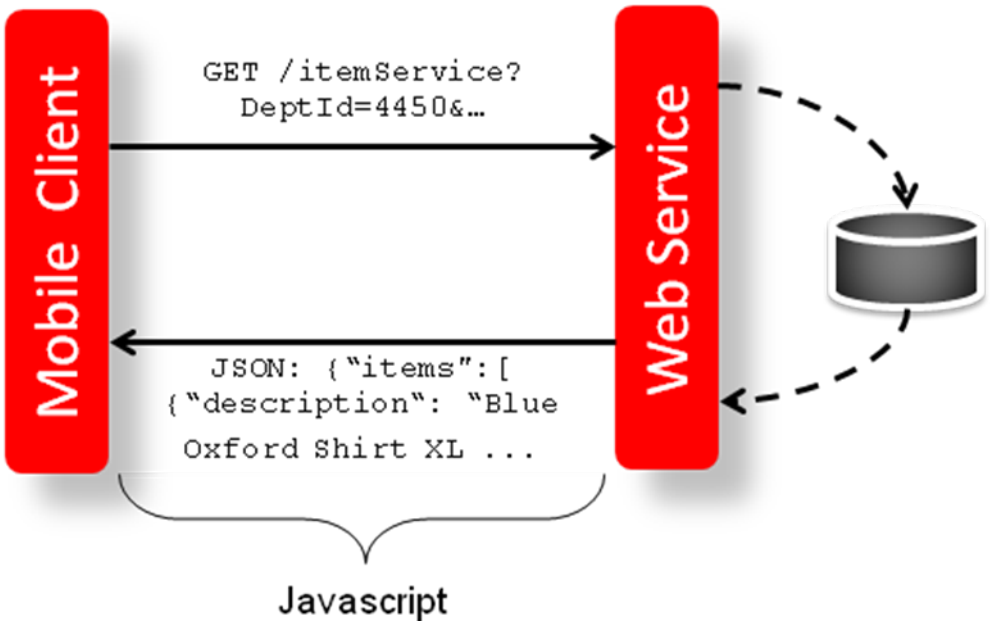
This chapter gives an overview about the RMS ReSTful Web service Implementation API designs used in the RMS environment and various functional attributes used in the APIs.

Retailers can access backend functionality by calling the application's ReSTful Web services. For more information on ReST architectural style applied for building Web services, access the following URL:

http://www.oracle.com/technetwork/articles/javase/index-137171.html.

Introduction

The RMS ReSTful Web services are based on a mobile application defined by Oracle Retail. The services are designed to support the mobile application functionality.



Mobile Client and Web Services integration through Javascript

Other Uses

The main objective of the ReSTful Web service is to support mobile applications, some of the services functionalities may not be useful for general use.The services are used for client's custom mobile application.

The ReSTful Web services Java code cannot be customized, but the MBL PL/SQL functions and object types can be customized for a client use. In addition, the client uses the RSE tool to create their own custom services in place of the ReSTful Web services.

Using ReSTful Web Service

The services should not be used during the restricted batch window.

Common Characteristics of Retail Application ReSTful Web Services

Deployment

A retail application packages its ReST services as part of the application's Enterprise Archive (EAR) file. The services are packaged as a Web Archive (WAR) within the EAR file.

The ReST Web services are installed by default. The RMS EAR file includes only the RMS ReST services.

Security

Services are secured using J2EE-based security model.

- **Realm-based User Authentication:** This verifies users through an underlying Realm. The username and password is passed using HTTP basic authentication.
- **Role-based Authorization:** This assigns users to roles; you can either grant or restrict access to the resource/service. The authorization to the ReSTful Web services is static and cannot be reassigned to other roles during post installation.

  The following table gives the Role-Name and Principal-Name; the Role is associated with ReSTful Web services and should be added to the Enterprise LDAP:

ROLE-NAME	PRINCIPAL-NAME
DATA_STEWARD_MANAGER	DATA_STEWARD_MANAGER_JOB

All enterprise roles defined in the table are mapped in web.xml and weblogic.xml of the ReST Service web application.

- The communication between the server and the client is encrypted using one way SSL. In a non-SSL environment, the encoding defaults to BASE-64 so it is highly recommended that these ReST services are configured to be used in production environments secured with SSL connections.
- RMS user data security is implemented in the APIs. The application user ID should be added to the RMS SEC_USER.APP_USER_ID table then associated to the appropriate group in SEC_USER _GROUP table.

Standard Request and Response Headers

Retail Application ReSTful Web services have the following standard HTTP headers:

```
Accept: application/xml or application/JSON
Accept-Version: 14.1 (service version number)
Accept-Language: en-US,en;q=0.8
```

Depending on the type of the operation or HTTP method, the corresponding response header is updated in the HTTP response with the following codes:

- `GET/READ : 200`
- `PUT/CREATE : 201 created`
- `POST/UPDATE : 204`
- `DELETE : 204`

Standard Error Response

Example response payload in case of service error is shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<messagesRDOes>
  <messagesRDO>
     <message>REST Service Version Mismatch</message>
     <messageType>ERROR</messageType>
     <status>BAD_REQUEST</status>
 </messagesRDO>
</messagesRDOes>
```

- Message: The error message - translated.
- MessageType: Value of 'ERROR' is returned.
- Status: For a bad request or error, the status is BAD_REQUEST.
- The http error code for an error response is 400.

URL Paths

The following links provide access to the Web services:

- The RMS ReSTful Web services Javadoc are available at:
  `http://<host:port>/RMSService`
- The RMS ReSTful Web services's WADL file is available at:
  `http://<host:port>/RMSService/services/private/application.wadl`
- The RMS ReSTful Web services are available at:
  `http://<host:port>/RMSService/services/private</service>`

Date Format

All input date and output date fields must be in long format.

Paging

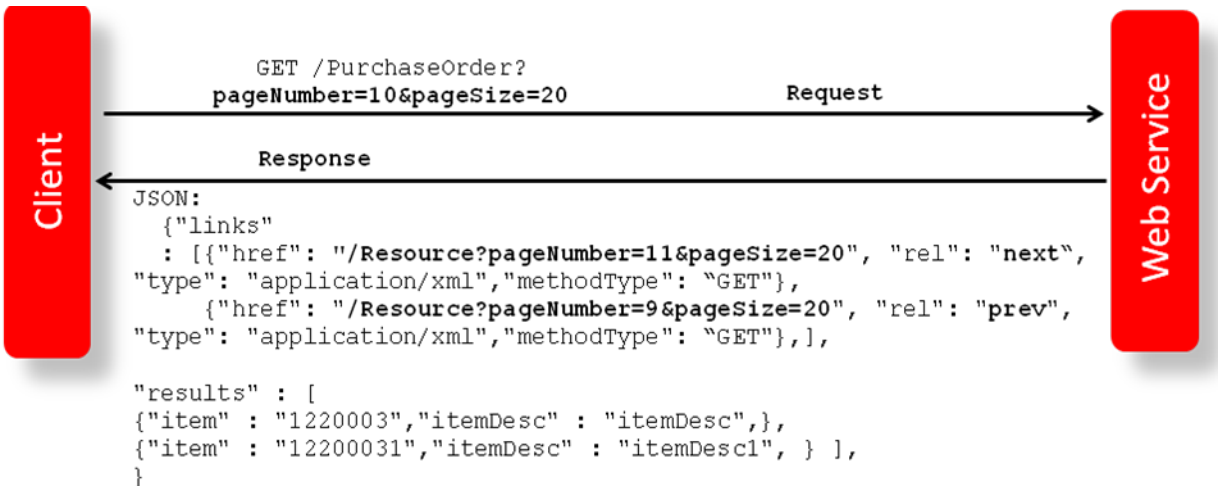
Some of the RMS ReSTful Web services have the potential to bring back a large number of records and therefore these services are equipped to segmenting the result into pages. The page number to retrieve and the size of the page are added as input parameters to all the paged services.

Each paged result includes the following information:

- **Total Record Count:** Displays the number of all records matching the service input criteria.
- **Next Page URL**: Shows the service URL with same input parameters, but with the pageNumber plus 1, when more records exists.
- **Previous Page URL**: Shows the service URL with same input parameters and the pageNumber input value minus 1, when page number is not 1.

Next or previous page URL is not provided when:

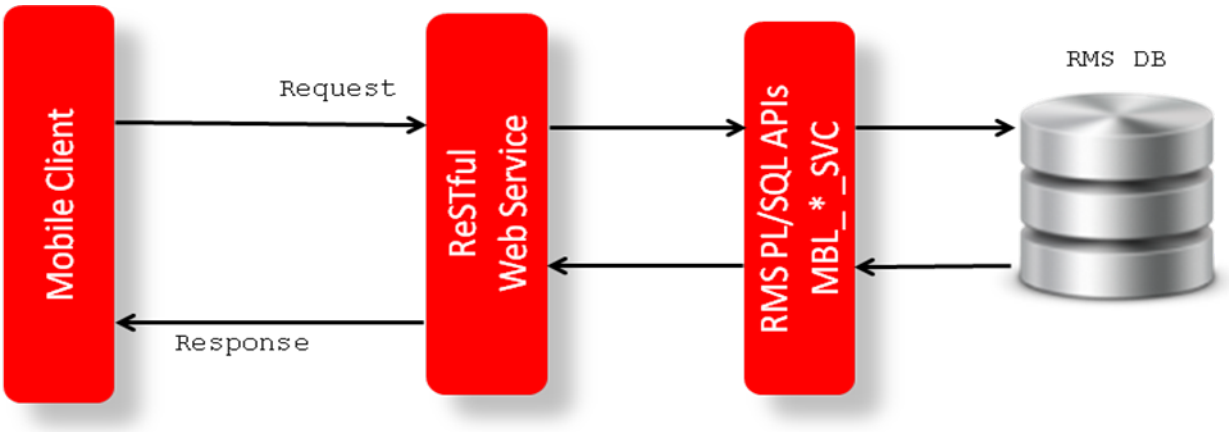
- No records are returned.
- Previous page is not returned, when the page number is 1.
- Next page is not returned, when the record reaches the last page.



Javascript for Paging Information in RMS Web Services

Web Service APIs Process Flow

The diagram shows the Web service API process flow.



Web Service APIs Process Flow

List of ReSTful Web Services

RMS Common Services

Functional Area
Foundation

Business Overview
The primary role of this service is to provide access to cross-functional RMS data.

Vdate

Business Overview
Retrieve RMS Vdate.

Service Type
Get

ReST URL
`/Common/vDate`

Input Parameters
NA

Output
Vdate in long format

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
PERIOD	Yes	No	No	No

Procurement Unit Options

Business Overview
Retrieve RMS's Procurement Unit Options.

Service Type
Get

ReST URL
`/Common/POSysOps`

Input Parameters
NA

Output

Values of the following columns:

- BACKPOST_RCA_RUA_IND
- BILL_TO_LOC
- CALC_NEGATIVE_INCOME
- COPY_PO_CURR_RATE
- COST_LEVEL
- CREDIT_MEMO_LEVEL
- DEAL_AGE_PRIORITY
- DEAL_LEAD_DAYS
- DEAL_TYPE_PRIORITY
- DEPT_LEVEL_ORDERS
- EDI_COST_OVERRIDE_IND
- EXPIRY_DELAY_PRE_ISSUE
- GEN_CONSIGNMENT_INVC_FREQ
- GEN_CON_INVC_ITM_SUP_LOC_IND
- LATEST_SHIP_DAYS
- ORD_APPR_CLOSE_DELAY
- ORD_APPR_AMT_CODE
- ORD_AUTO_CLOSE_PART_RCVD_IND
- ORD_PART_RCVD_CLOSE_DELAY
- ORDER_BEFORE_DAYS
- ORDER_EXCH_IND
- OTB_SYSTEM_IND
- RCV_COST_ADJ_TYPE
- RECLASS_APPR_ORDER_IND
- REDIST_FACTOR
- SOFT_CONTRACT_IND
- WAC_RECALC_ADJ_IND

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
PROCUREMENT_UNIT_OPTIONS	Yes	No	No	No

Functional Config Options

Business Overview

Retrieve RMS's Functional Config Options.

Service Type

Get

ReST URL

```
/Common/FuncSysOps
```

Input Parameters

NA

Output

Values of the following columns:

- CONTRACT_IND
- ELC_IND
- IMPORT_IND
- ORG_UNIT_IND
- SUPPLIER_SITES_IND

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
FUNCTIONAL_CONFIG_OPTIONS	Yes	No	No	No

Inventory Movement Unit Options

Business Overview

Retrieve RMS's Inventory Movement Unit Options.

Service Type

Get

ReST URL

```
/Common/InvMovSysOps
```

Input Parameters

NA

Output

Values of the following columns:

- ALLOC_METHOD
- APPLY_PROF_PRES_STOCK
- AUTO_RCV_STORE
- CLOSE_OPEN_SHIP_DAYS
- COST_MONEY
- COST_OUT_STORAGE
- COST_OUT_STORAGE_MEAS
- COST_OUT_STORAGE_UOM
- COST_WH_STORAGE
- COST_WH_STORAGE_MEAS
- COST_WH_STORAGE_UOM
- DEFAULT_ALLOC_CHRG_IND
- DEFAULT_ORDER_TYPE
- DEFAULT_SIZE_PROFILE
- DEPT_LEVEL_TRANSFERS
- DISTRIBUTION_RULE
- DUPLICATE_RECEIVING_IND
- INCREASE_TSF_QTY_IND
- INTERCOMPANY_TRANSFER_BASIS
- INV_HIST_LEVEL
- LOC_ACTIVITY_IND
- LOC_DLVRY_IND
- LOOK_AHEAD_DAYS
- MAX_SCALING_ITERATIONS
- MAX_WEEKS_SUPPLY
- ORD_WORKSHEET_CLEAN_UP_DELAY
- RAC_RTV_TSF_IND
- REJECT_STORE_ORD_IND
- REPL_ORDER_DAYS
- RTV_NAD_LEAD_TIME
- RTV_UNIT_COST_IND
- SHIP_RCV_STORE
- SHIP_RCV_WH
- STORAGE_TYPE
- STORE_PACK_COMP_RCV_IND
- WF_DEFAULT_WH
- TARGET_ROI
- TSF_AUTO_CLOSE_STORE
- TSF_AUTO_CLOSE_WH
- TSF_CLOSE_OVERDUE
- SIM_FORCE_CLOSE_IND
- TSF_FORCE_CLOSE_IND
- TSF_OVER_RECEIPT_IND
- TSF_MD_STORE_TO_STORE_SND_RCV
- TSF_MD_STORE_TO_WH_SND_RCV
- TSF_MD_WH_TO_STORE_SND_RCV
- TSF_MD_WH_TO_WH_SND_RCV
- TSF_PRICE_EXCEED_WAC_IND
- SS_AUTO_CLOSE_DAYS
- WS_AUTO_CLOSE_DAYS
- SW_AUTO_CLOSE_DAYS
- WW_AUTO_CLOSE_DAYS
- WF_ORDER_LEAD_DAYS
- WH_CROSS_LINK_IND
- WRONG_ST_RECEIPT_IND

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
INV_MOVE_UNIT_OPTIONS	Yes	No	No	No

Currencies

Business Overview

Retrieve RMS's Currencies table records.

Service Type

Get

ReST URL

`/Common/Currencies`

Input Parameters

NA

Output

Fetches the following columns for all the currencies:

- Currency Code
- Currency Description
- Currency Cost Format
- Currency Retail Format
- Currency Cost Decimal
- Currency Retail Decimal

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
CURRENCIES	Yes	No	No	No

Create Purchase Order Services

Functional Area

Procurement

Business Overview

The primary role of this service is to create purchase orders and send it to RMS.

Order Number

Business Overview

Retrieves the next order number from RMS.

Service Type
Get

ReST URL
`/PurchaseOrders/order/id`

Input Parameters
NA

Output
Order Number

Table Impact
NA

Terms

Business Overview
Retrieves all valid terms; valid terms are enabled flag set to Yes and within the start and end active date.

Service Type
Get

ReST URL
`/PurchaseOrders/supplier/terms`

Input Parameters
NA

Output
Values of the following columns:
Terms
Terms Code
Terms Description

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
TERMS_HEAD	Yes	No	No	No
TERMS_DETAIL	Yes	No	No	No

Search Supplier

Business Overview

Supplier search can be, by entering either full or partial supplier site ID (numeric) or by a full or partial supplier site description in the search string.

Returned suppliers are constrained by the following criteria:

- Only active supplier sites are returned.
- When items are sent as input, then only supplier sites that are common amongst the items are returned.
- When locations are sent as input, then only suppliers that are valid for the Org Units associated with the input locations are returned.

Service Type

Get

ReST URL

```
/PurchaseOrders/supplier?supplierSearchString={supplierSearchStri
ng}&locations={locations}&items={items}&pageSize={pageSize}&pageN
umber={pageNumber}
```

Input Parameters

Parameter Name	Required	Description
SupplierSearchString	Yes	Search string for Supplier's ID or Name.
Item	No	Comma Separated values for items.
Locations	No	Comma Separated values for locations.
PageSize	No	Maximum number of suppliers to retrieve per page.
PageNumber	No	Result page to retrieve.

Output

Values of the following columns:

Supplier

Supplier Name

Supplier Currency

Terms

Default Item Lead Time

Supplier Item Table

- Item
- Origin Country Id
- Lead Time

Supplier Item Location Table

- Item
- Location
- Pickup Lead Time

Total Record Count

Next Page URL

Previous Page URL

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_SUPPLIER	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
ITEM_SUPP_COUNTRY_LOC	Yes	No	No	No
STORE	Yes	No	No	No
SUPS	Yes	No	No	No
V_SUPS	Yes	No	No	No
WH	Yes	No	No	No

Load Supplier

Business Overview

Loading supplier Web service allows a user to refresh the selected supplier records.

Service Type

Get

ReST URL

```
/PurchaseOrders/supplier/load?suppliers={suppliers}&locations={locations}&items={items}
```

Input Parameters

Parameter Name	Required	Description
Supplier	Yes	Supplier's ID.
Item	No	Comma Separated values for items.
Locations	No	Comma Separated values for locations.

Output

Values of the following columns:
Supplier
Supplier Name
Supplier Currency
Terms
Default Item Lead Time
Supplier Item Table

- Item
- Origin_country Id
- Lead Time

Supplier Item Location Table

- Item
- Location
- Pickup Lead Time

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ITEM_SUPPLIER	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
ITEM_SUPP_COUNTRY_LOC	Yes	No	No	No
STORE	Yes	No	No	No
SUPS	Yes	No	No	No
V_SUPS	Yes	No	No	No
WH	Yes	No	No	No

Search Items

Business Overview

This service retrieves items applicable for Purchase Order. Item can be searched by either Item or VPN. Enter an item number, a partial item description, or a VPN in the search string.

1. When search type is ITEM, the search string can be an item number, a partial item number, an item description, or partial item description
2. When search type is VPN, the search string can be a VPN or partial VPN.

The items returned are constrained by the following criteria:

- Approved status.
- Transaction-level items.
- Orderable items.
- Pack items with Order Type as Each are filtered out.
- Only items belonging to Normal Merchandise Purchase Type as Department are retuned.
- When a supplier is sent as input then:
  - Only items supplied by the input supplier are returned.
  - The item information is based on the Item/Supplier/Primary Origin Country.
- When supplier is not sent as input, then item information is based on the primary supplier and primary origin country.
- If the system_options.dept_level_orders is set to"Y" and the Department ID is sent as input, then only the input department items are returned.
- Items set for deletion are filtered out.

Service Type

Get

ReST URL

```
/PurchaseOrders/item?itemSearchType={itemSearchType}&searchString
={searchString}&dept={dept}&supplier={supplier}&locations={locati
ons}&pageSize={pageSize}&pageNumber={pageNumber}
```

Input Parameters

Parameter Name	Required	Description	Valid values
itemSearchType	Yes	Search Type item or VPN.	ITEM, VPN
searchString	Yes	Search string for items Id or Name.	NA
dept	No	Selected items' department ID.	NA
supplier	No	Selected Supplier ID.	NA
Locations	No	Comma Separated values for selected locations' ID.	NA
PageSize	No	Maximum number of items to retrieve per page.	NA
PageNumber	No	Result page to retrieve.	NA

Output

Values of the following columns:

Item

Item Description

Department

Supplier

Origin Country Id

Supplier Pack Size

Unit Cost

Supplier Currency

Base Unit Retail

Base Retail Currency

Base Retail UOM

Item Image URL

Locations Table

- Location
- Location Type
- Item Location Status
- Unit Retail
- Unit Retail Currency
- Unit Retail UOM

Total Record Count

Next Page URL

Previous Page URL

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
DAILY_PURGE	Yes	No	No	No
DEPS	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
ITEM_IMAGE	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
ITEM_SUPPLIER	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
MV_CURRENCY_CONVERSION_RATES	Yes	No	No	No
RPM_MERCH_RETAIL_DEF_EXPL	Yes	No	No	No
RPM_ZONE	Yes	No	No	No
V_ITEM_MASTER	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
V_SUPS	Yes	No	No	No
WH	Yes	No	No	No

Load Items

Business Overview

The primary use of loading items Web service is to refresh already selected PO items records.

Service Type

Get

ReST URL

```
/PurchaseOrders/item/load?item=item&supplier={supplier}&locations
={locations}
```

Input Parameters

Parameter Name	Required	Description
Items	Yes	Comma Separated values for selected items' ID.
Supplier	No	Selected Supplier ID.
Locations	No	Comma Separated values for selected locations' ID.

Output

Values of the following columns:
Item
Item Description
Department
Supplier
Origin Country Id
Supplier Pack Size
Unit Cost
Supplier Currency
Base Unit Retail
Base Retail Currency
Base Retail UOM
Item Image URL
Locations Table

- Location
- Location Type
- Item Location Status
- Unit Retail
- Unit Retail Currency
- Unit Retail UOM

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
DAILY_PURGE	Yes	No	No	No
DEPS	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
ITEM_IMAGE	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
ITEM_SUPPLIER	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
MV_CURRENCY_CONVERSION_RATES	Yes	No	No	No
RPM_MERCH_RETAIL_DEF_EXPL	Yes	No	No	No
RPM_ZONE	Yes	No	No	No
V_ITEM_MASTER	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
V_SUPS	Yes	No	No	No
WH	Yes	No	No	No

Search Locations

Business Overview

The Web service enables location search applicable for PO. Location can be searched by either 'S'tore or 'W'arehouse. Enter a location number, a partial location number, a location description, or a partial location description in the search string.

The locations returned are constrained by the following criteria:

- Only stockholding locations are returned.
- When search type is Warehouse then:
  - Only virtual warehouses are returned.
  - Internal finishers are filtered out.
- When search type is store then only the following stores are returned:
  - Company stores.
  - Open stores.
- When system_options.org_unit_ind is set as 'Y' then:
  - When supplier is sent as input then only locations with same org_unit_id are returned.
  - When Org Unit ID is sent as input then only locations with same org_unit_id are returned.

Service Type

Get

ReST URL

```
/PurchaseOrders/location?locationType={locationType}&searchString
={searchString}&supplier={supplier}&orgUnitId={orgUnitId}&pageSiz
e={pageSize}&pageNumber={pageNumber}
```

Input Parameters

Parameter Name	Required	Description	Valid values
LocationType	Yes	Location type Store or warehouse.	S, W
SearchString	Yes	Search string for locations Id or Name.	NA
Supplier	No	Selected Supplier ID.	NA
OrgUnitId	No	Selected locations' Org unit ID.	NA
PageSize	No	Maximum number of locations to retrieve per page.	NA
PageNumber	No	Result page to retrieve.	NA

Output

Values of the following columns:
Location
Location Type
Location Name
Location Currency
Org Unit Id
Total Record Count
Next Page URL
Previous Page URL

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
PARTNER_ORG_UNIT	Yes	No	No	No
V_STORE	Yes	No	No	No
V_WH	Yes	No	No	No
WH	Yes	No	No	No

Load Locations

Business Overview

This Web service allows the user to refresh already selected PO locations records.

Service Type

Get

ReST URL

```
/PurchaseOrders/location/load?locations={locations}&supplier={supplier}
```

Input Parameters

Parameter Name	Required	Description
Locations	Yes	Comma Separated values for selected locations' ID.
Supplier	No	Selected Supplier ID.

Output

Values of the following columns:
Location
Location Type
Location Name
Location Currency
Org Unit Id

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
PARTNER_ORG_UNIT	Yes	No	No	No
V_STORE	Yes	No	No	No
V_WH	Yes	No	No	No
WH	Yes	No	No	No

Create Purchase Order

Business Overview

This Web service calls the existing RMS XOrder API directly with input parameters. For more information on RMS XOrder API, see Store Order Subscription API and PO Subscription API sections.

Service Type

Post

ReST URL

```
/PurchaseOrders
```

Input Parameters

```
Example json RDO input:
      {
         "links" : [ ],
          "itemRDOs" : [ {
            "links" : [ ],
           "item" : null,
           "location" :null,
            "unitCost" : null,
            "referenceItem" : null,
            "originCountryId" : null,
            "suppPackSize" : null,
            "qtyOrdered" : null,
            "locationType" : null,
            "cancelInd" : null,
            "reInstateInd" : null,
            "hyperMediaContent" : {
              "linkRDO" : [ ]
            }
          } ],
          "orderNo" : null,
          "supplier" : null,
          "currencyCode" : null,
          "terms" : null,
          "notBeforeDate" : null,
          "notAfterDate" : null,
          "status" : "A",
```

```
    "writtenDate" : null,
    "origInd" : null,
    "user_id" : null,
    "dept" : null,
    "exchangeRate" : null,
    "includeOnOrdInd" : null,
    "ediPoInd" : null,
    "preMarkInd" : null,
    "comment" : null,
    "otbEowDate" : null,
    "hyperMediaContent" : {
      "linkRDO" : [ ]
    }
  }
```

Output

NA

Table Impact

For more information on RMS XOrder API, see Store Order Subscription API and PO Subscription API sections.

Create Inventory Transfer Services

Functional Area

Inventory Movement

Business Overview

The primary role of these services is to create transfers and send them to RMS.

Transfer Number

Business Overview

Retrieves the next transfer number from RMS.

Service Type

Get

ReST URL

`/Transfer/TransferId`

Input Parameters

No input

Output

Transfer Number

Table Impact

NA

Search Items

Business Overview

This service retrieves items applicable for inventory transfer. Item can be searched either by Item or VPN. To search the item, enter an item number, a partial item description, or a VPN in the search string.

- When search type is ITEM, the search string can be an item number, a partial item number, an item description, or partial item description. In this case, the query returns all items which match the item description or partial description, or which match the item number entered.
- When search type is VPN, the search string can be a VPN or partial VPN, the API should return all items with that VPN.

  The items returned are constrained by the following criteria:
  - Approved status.
  - Transaction-level items.
  - Inventory items.
- When From Location is sent as an input, then only the following items are returned:
  - With available inventory at the From Location.
  - Packs with Receive as Type as Each are filtered out when, from location is a virtual warehouse.
- If the System Option for DEPT_LEVEL_TRANSFERS is set as"Y" and a Department ID is sent as input, then only the input department items are returned.

Service Type

Get

ReST URL

```
/Transfer/item?itemSearchType={itemSearchType}&searchString={sear
chString}&dept={dept}&fromLocation={fromLocation}&pageSize={pageS
ize}&pageNumber={pageNumber}
```

Input Parameters

Parameter Name	Required	Description	Valid values
itemSearchType	Yes	Search type item or VPN.	ITEM, VPN
searchString	Yes	Search string for items ID or Name.	NA
dept	No	Selected items' department ID.	NA
fromLocation	No	Selected from location ID.	NA
PageSize	No	Maximum number of items to retrieve per page.	NA
PageNumber	No	Result page to retrieve.	NA

Output

Values of the following columns:

- Item
- Item Description
- Department
- Available Quantity --only populated when from loc is sent
- Average Cost --only populated when from loc is sent
- Unit Retail --only populated when from loc is sent
- Currency Code --only populated when from loc is sent
- Standard UOM
- Supplier Pack Size
- Inner Pack Size
- Item Image URL
- Total Record Count
- Next Page URL
- Previous Page URL

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ALLOC_HEADER	Yes	No	No	No
ALLOC_DETAIL	Yes	No	No	No
DAILY_PURGE	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
ITEM_LOC_SOH	Yes	No	No	No
ITEM_IMAGE	Yes	No	No	No
ITEM_SUPPLIER	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
ORDHEAD	Yes	No	No	No
STORE	Yes	No	No	No
V_ITEM_MASTER	Yes	No	No	No
WH	Yes	No	No	No

Load Items

Business Overview

Load items service allows the user to refresh item records information for already selected items.

Service Type

Get

ReST URL

```
/Transfer/item/load?items={items}&fromLocation={fromLocation}
```

Input Parameters

Parameter Name	Required	Description
items	Yes	Comma Separated values for selected items' ID.
fromLocation	No	Selected from location ID.

Output

Values of the following columns:

- Item
- Item Description
- Department
- Available Quantity --only populated when from loc is sent
- Average Cost --only populated when from loc is sent
- Unit Retail --only populated when from loc is sent
- Currency Code --only populated when from loc is sent
- Standard UOM
- Supplier Pack Size
- Inner Pack Size
- Item Image URL

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ALLOC_HEADER	Yes	No	No	No
ALLOC_DETAIL	Yes	No	No	No
DAILY_PURGE	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
ITEM_LOC_SOH	Yes	No	No	No
ITEM_IMAGE	Yes	No	No	No
ITEM_SUPPLIER	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
ORDHEAD	Yes	No	No	No
STORE	Yes	No	No	No
V_ITEM_MASTER	Yes	No	No	No
WH	Yes	No	No	No

Search From Location

Business Overview

This service retrieves locations applicable for inventory transfer. Location can be searched by either 'S'tore or 'W'arehouse. Then enter a location number, a partial location number, a location description, or a partial location description in the search string.

The locations returned are constrained by the following criteria:

- When search type is warehouse only virtual warehouses are returned.
- Only stockholding location.
- When search type is store then only open stores are returned.
- When items are sent as input then only locations with available inventory are returned.
- When To Location is sent as input then:
    - It cannot be the same as the To Location.
    - When transfer type is Manual Requisition, then only locations with the same Transfer Entity/Set of Books as the To Location are returned in the search results.
    - When the transfer type is Intercompany, then only locations with a different Transfer Entity/Set of Books to the To Location are returned in the search results.
    - Only locations in the same transfer zone are returned in the search results.

Service Type

Get

ReST URL

```
/Transfer/fromLocation?locationType={locationType}&searchString={
searchString}&tsfType={tsfType}&toLocation={toLocation}&items={it
ems}&pageSize={pageSize}&pageNumber={pageNumber}
```

Input Parameters

Parameter Name	Required	Description	Valid values
LocationType	Yes	Location type Store or warehouse	S, W
SearchString	Yes	search string for locations Id or Name	NA
tsfType	Yes	Transfer type	IC, MR
toLocation	No	Selected to location ID	NA
items	No	Comma Separated values for selected items	NA
PageSize	No	Maximum number of locations to retrieve per page	NA
PageNumber	No	Result page to retrieve	NA

Output

Values of the following columns:

- Location
- Location Type
- Location Name
- Location Currency
- Entity
- Entity Description
- Location Item Table
    - Item
    - Available Quantity
    - Average Cost
    - Unit Retail
    - Currency Code
- Total Record Count
- Next Page URL
- Previous Page URL

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ALLOC_HEADER	Yes	No	No	No
ALLOC_DETAIL	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
ITEM_LOC_SOH	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
MV_LOC_SOB	Yes	No	No	No
ORDHEAD	Yes	No	No	No
ORG_UNIT	Yes	No	No	No
STORE	Yes	No	No	No
TRANSFER_LOC	Yes	No	No	No
TSF_ENTITY	Yes	No	No	No
V_STORE	Yes	No	No	No
V_TRANSFER_FROM_LOC	Yes	No	No	No
V_TRANSFER_TO_LOC	Yes	No	No	No
V_WH	Yes	No	No	No

Search To Location

Business Overview

This service retrieves locations applicable for inventory transfer. Location can be searched by either 'S'tore or 'W'arehouse. Then enter a location number, a partial location number, a location description, or a partial location description in the search string.

The locations returned are constrained by the following criteria:

- When search type is warehouse only virtual warehouses are returned.
- Internal finishers are filtered out.
- Only stockholding location.
- When search type is Store then only open stores are returned.
- When items are sent as input then only locations with available inventory are returned.
- When From Location is sent as input then:
    - To Location cannot be the same as the From Location.
    - When Transfer Type is set as a manual request, then only locations with the same Transfer Entity/Set of Books as the From Location are returned in the search results.
    - When the Transfer Type is Intercompany, then only locations with a different Transfer Entity/Set of Books to the From Location are returned in the search results.
    - Only locations in the same transfer zone are returned in the search results.

Service Type

Get

ReST URL

```
/Transfer/toLocation?locationType={locationType}&searchString={searchString}&tsfType={tsfType}&fromLocation={fromLocation}&pageSize={pageSize}&pageNumber={pageNumber}")
```

Input Parameters

Parameter Name	Required	Description	Valid values
LocationType	Yes	Location type Store or warehouse	S, W
SearchString	Yes	search string for locations Id or Name	NA
tsfType	Yes	Transfer type	IC, MR
fromLocation	No	Selected from location ID	NA
PageSize	No	Maximum number of locations to retrieve per page	NA
PageNumber	No	Result page to retrieve	NA

Output

Values of the following columns:

- Location
- Location Type
- Location Name
- Location Currency
- Entity
- Entity Description
- Location Item Table
    - Item
    - Available Quantity
    - Average Cost
    - Unit Retail
    - Currency Code
- Total Record Count
- Next Page URL
- Previous Page URL

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ALLOC_HEADER	Yes	No	No	No
ALLOC_DETAIL	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
ITEM_LOC_SOH	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
MV_LOC_SOB	Yes	No	No	No
ORDHEAD	Yes	No	No	No
ORG_UNIT	Yes	No	No	No
STORE	Yes	No	No	No
TRANSFER_LOC	Yes	No	No	No
TSF_ENTITY	Yes	No	No	No
V_STORE	Yes	No	No	No
V_TRANSFER_FROM_LOC	Yes	No	No	No
V_TRANSFER_TO_LOC	Yes	No	No	No
V_WH	Yes	No	No	No

Load Locations

Business Overview

Load locations Web service allows user to refresh selected locations records.

Service Type

Get

ReST URL

```
/Transfer/loadLocations?fromLocation={fromLocation}&toLocation={toLocation}
```

Input Parameters

Parameter Name	Required	Description
FromLocation	No	Selected from location ID.
ToLocation	No	Selected to location ID.

Output

Values of the following columns:

- Location
- Location Type
- Location Name
- Location Currency
- Entity
- Entity Description
- Location Item Table
    - Item
    - Available Quantity
    - Average Cost
    - Unit Retail
    - Currency Code

Table Impact

TABLE	SELECT	INSERT	UPDATE	DELETE
ALLOC_HEADER	Yes	No	No	No
ALLOC_DETAIL	Yes	No	No	No
ITEM_LOC	Yes	No	No	No
ITEM_LOC_SOH	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
MV_LOC_SOB	Yes	No	No	No
ORDHEAD	Yes	No	No	No
ORG_UNIT	Yes	No	No	No
STORE	Yes	No	No	No
TRANSFER_LOC	Yes	No	No	No
TSF_ENTITY	Yes	No	No	No
V_STORE	Yes	No	No	No

TABLE	SELECT	INSERT	UPDATE	DELETE
V_TRANSFER_FROM_LOC	Yes	No	No	No
V_TRANSFER_TO_LOC	Yes	No	No	No
V_WH	Yes	No	No	No

Create Transfer

Business Overview

The Web service calls the existing RMS XTSF API directly with input parameters. For more information on RMS XTSF API, see Store Order Subscription API and Transfer Subscription API sections.

Service Type

Post

ReST URL

`/Transfer`

Input Parameters

```
Example json RDO input:
{
    "links" : [ ],
     "tsfdtlRDOs" : [ {
       "links" : [ ],
       "item" : null,
       "tsfQty" : null,
       "suppPackSize" : null,
       "invStatus" : null,
       "unitCost" : null,
       "hyperMediaContent" : {
         "linkRDO" : [ ]
       }
     } ],
     "tsfNo" : null,
     "fromLocType" : null,
     "fromLoc" : null,
     "toLocType" : null,
     "toLoc" : null,
     "deliveryDate" : null,
     "dept" : null,
     "routingCode" : null,
     "freightCode" : null,
     "tsfType" : null,
     "status" : null,
     "userId" : null,
     "commentDesc" : null,
     "contextType" : null,
```

```
    "contextValue" : null,
    "hyperMediaContent" : {
      "linkRDO" : [ ]
    }
  }
```

Output

NA

Table Impact

For more information on RMS XTSF API, see Store Order Subscription API and Transfer Subscription API sections.

A

Appendix: RIB_XML

Introduction

The RIB_XML Procedural Language/Structured Query Language (PL/SQL) package contains a set of utilities to make the generation and parsing of XML documents easier. It is based on Oracle's XML Developer's Kit (XDK), and is designed to support application-specific Application Programming Interfaces (APIs) that read and write Extensible Markup Language (XML) messages.

The Oracle XML Developer's Kit (XDK) contains an XML parser and an implementation of the World Wide Web Consortium (W3C) Document Object Model (DOM). The RIB_XML package provides streamlined access to Oracle's APIs, allowing developers to quickly produce simple documents with a minimum of fuss.

The W3C DOM is an industry-standard set of data structures and APIs that allow developers to access and manipulate the contents of an XML document. The DOM is full-featured and is portable to a variety of platforms, but is generally considered unwieldy and over-structured for most people's needs. For our APIs, a more simplified model is used. The RIB_XML package attempts to hide most of the complexities of the W3C DOM and Oracle's implementation of the DOM behind a series of functions that take care of things like element generation and text node management.

In our simplified DOM, we only deal with element nodes. Documents, text nodes, and all the type-casting that one normally associates with the W3C DOM's everything-is-a-node philosophy are taken care of by the RIB_XML package. The utility functions in this package should be sufficient for nearly all documents, but since we are working with the standard structures and APIs, the full set of functionality is available if needed.

Generating XML Documents

> **Note**: For a faster but more restrictive method of writing XML documents, you may want to use RIB_SXW.

An XML document is created by building a DOM tree and writing the contents of the tree to a CLOB or string. A new document is started by calling RIB_XML.newRoot(), which initializes an element and designates it the root of the DOM tree.

Adding Elements to the Tree

Elements other than root are added to the tree with the RIB_XML.addElement() procedure. If an element contains only text, it can be set by providing the optional value argument to addElement(). If no value is provided, the element will be empty. After calling addElement() with no value for the node, you can call addElementContents() to set the value. There is also a function form of AddElement() which returns the element it created, so that it can be further manipulated (e.g., having elements added to it in turn).

In addition to using addElement() to add text values to a DOM tree, you can also use addDateElement() to add dates in a format-independent manner. The addDateElement() function takes the same parent, name, and value arguments asaddElement, but also adds an inclTime argument. This Boolean flag tells the package to include the timestamp (hour, minute, and second) with the date. If it is false, no

timestamp is added, and only year, month, and day is added. The parameter is false by default.

The key to using addElement-type procedures and functions is to realize that each takes a parent element argument, and that the new element is added as the last child of this parent. As a result, when adding elements to the tree order counts. Furthermore, any message can be built with only addElement by adding elements in the order given by the message type format specification.

When the document is complete, it is written to the target CLOB with the RIB_XML.writeRoot() function. After a document is done with, resources are freed with the RIB_XML.freeRoot() procedure.

Example 1: Generating an XML document

This PL/SQL function:

```
create or replace procedure example1 is
   root    xmldom.DOMElement;
   message clob;
begin
   dbms_lob.createtemporary(message, TRUE);
   if rib_xml.newRoot(root, 'foobar') = FALSE then
      return;
   end if;
   --
   rib_xml.addElement(root, 'whine', 'Do I have to?');
   rib_xml.addElement(root, 'moan');
   rib_xml.addDateElement( root, 'foodate', sysdate, true );
   --
   if rib_xml.writeRoot(root, message, true) = FALSE then
      return;
   end if;
end;
/
show errors
```

Returns this XML document:

```
<foobar xmlns="http://www.oracle.com/retail/integration/payload/foobar"
xmlns:ribdate="http://www.oracle.com/retail/integration/payload/RIBDate"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.oracle.com/retail/integration/payload/foobar
http://mspdev81:7777/rib-func-artifact/payload/xsd/foobar.xsd
http://www.oracle.com/retail/integration/payload/RIBDate http://mspdev81:7777/rib-
func-artifact/payload/xsd/RIBDate.xsd">
  <whine>Do I have to?</whine>
  <moan/>
  <foodate>
    <ribdate:year>2007</ribdate:year>
    <ribdate:month>11</ribdate:month>
    <ribdate:day>30</ribdate:day>
    <ribdate:hour>10</ribdate:hour>
    <ribdate:minute>41</ribdate:minute>
    <ribdate:second>06</ribdate:second>
  </foodate>
</foobar>
```

Reading XML documents

An XML document is read from a CLOB and turned into a DOM tree, which a program can traverse or manipulate in whatever way it needs to.
The RIB_XML.readRoot() function returns the root element of the DOM tree. For any given element, the RIB_XML.getChild() function returns the child element with the given name.

A very common task is to get the text out of a particular child of a given element. This is done by RIB_XML.getChildText(), which is really just a convenience method that calls getChild() and extracts the text value from that.

Just as with adding elements, there are convenience functions available for handing dates. The getChildDate() function works much like getChildText(), and finds the date for a child element with the given name.

If you have a series of child element with the same name (e.g., multiple addresses for a supplier), use the RIB_XML.getChildren() function to get the whole list of children with the given name, and get a particular element out of the list with theRIB_XML.getListElement() function.

After all processing of a document is completed, any allocated resources are released by calling the freeRoot() function.

Example 2: Reading an XML document

This PL/SQL procedure:

```
create or replace procedure example2(data in clob) as
   root      xmldom.DOMElement;
   food      date;
begin
   root := rib_xml.readRoot(data, 'foobar');
   dbms_output.put_line( 'whine: ' ||
                         rib_xml.getChildText( root, 'whine' ) );
   food := rib_xml.getChildDate( root, 'foodate' );
   dbms_output.put_line( 'date: ' || to_char(food, 'YYYY/MM/DD HH24:MI:SS') );
   rib_xml.freeRoot( root );
end;
/
show errors
```

Prints these contents when provided with the XML document created in example 1:

```
whine: Do I have to?
date: 2007/11/30 10:41:06
```

Further Reading

For more information, see the comments in the spec for the RIB_XML package describing each function.

For more information on XML, see http://www.xml.com/, http://www.w3.org/XML/, http://xml.coverpages.org/index.html.

For information on W3C DOM, see http://www.w3.org/DOM/, http://www.w3schools.com/dom/default.asp, http://www.xml.com/pub/rg/DOM_Tutorials.

For information on XML utilities and APIs, see http://www.oracle.com/technetwork/index.html, http://www.oracle.com/technetwork/index.html.

The ideas for the RIB_XML package were taken from http://www.jdom.org/, an XML-handling API for Java.