Oracle® Retail Merchandising System

Operations Guide Addendum

Release 11.0.10

February 2007

ORACLE

Oracle® Retail Merchandising System Operations Guide Addendum, Release 11.0.10

Primary Author: Nathan Young

Contents

Preface

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

- Key system administration configuration settings
- Technical architecture

Audience

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

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

Related Documents

For more information, see the following documents in the Oracle Retail Merchandising System Release 11.0.10 documentation set:

- Oracle Retail Merchandising System Installation Guide
- Oracle Retail Merchandising System Release Notes
- Oracle Retail Merchandising System Data Model
- Oracle Retail Merchandising System User Guide
- Oracle Retail Merchandising System Online Help
- Oracle Retail Merchandising System Batch Schedule

Customer Support

- https://metalink.oracle.com

When contacting Customer Support, please provide:

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

Conventions

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

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

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

A hyperlink appears like this.

1
Introduction

Overview

This addendum to the Oracle Retail Merchandising System (RMS) 11 Operations Guide presents changes that have resulted from work completed during RMS 11.0.10 development and customer support.

RMS 11.0.10 can be configured to integrate with a time-phased inventory planning tool through the RIB. RMS can capture the data that a time-phased inventory planning tool requires and publish that data to the RIB. The following sections address the time-phased inventory planning integration:

- Modifications to RMS – Oracle Retail Predictive Application Server (RPAS) Interface
- Subscription Designs

The ProC batch designs included in this document have been updated based on reported defects.

2

Modifications to RMS – Oracle Retail Predictive Application Server (RPAS) Interface

Because RMS is the retailer's central merchandising transactional processing system, the system is the principle source of the foundation data needed in some of the Oracle Retail suite of products. This chapter includes information regarding RETL programs related to the RMS-RPAS interface.

Versions

Please note that the modifications that have been made to RMS code are specific to the following versions:

- RMS 11.0.10

RETL Program Overview for the RMS-Time-Phased Inventory Planning Tool Interface

This chapter summarizes the RETL program features utilized in the RMS Extractions (RMSE) for the RMS–time-phased inventory planning tool integration. Starting with RMS version 11, the RMS extract for a time-phased inventory planning tool is separate from the RMS extracts for RPAS. In prior RMS version, time-phased inventory planning tool and RPAS had common RETL extracts.

More installation information about the RETL tool is available in the latest RETL Programmer's Guide.

> **Note:** In this section, some examples refer to RETL programs that are not related to RMS or are related to other versions of RMS than this document addresses. Such examples are included for illustration purposes only.

Installation

Select a directory where you would like to install RMS ETL. This directory (also called MMHOME) is the location from which the RMS ETL files are extracted.

The following code tree is utilized for the RETL framework during the extractions, transformations, and loads and is referred to in this documentation.

```
<base directory (MMHOME)>
          /data
          /error
          /log
          /rfx
                /bookmark
                /etc
                /lib
                /schema
                /src
```

Configuration

RETL

Before trying to configure and run RMS ETL, install RETL version 11.3 or later, which is required to run RMS ETL. See the latest RETL Programmer's Guide for thorough installation information.

RETL user and permissions

RMS ETL is installed and run as the RETL user. Additionally, the permissions are set up as per the RETL Programmer's Guide. RMS ETL reads data, creates, deletes and updates tables. If these permissions are not set up properly, extractions fail.

Environment Variables

See the RETL Programmer's Guide for RETL environment variables that must be set up for your version of RETL. You will need to set MMHOME to your base directory for RMS RETL. This is the top level directory that you selected during the installation process (see the section, 'Installation', above). In your .kshrc, you should add a line such as the following:

```
export MMHOME=<base directory for RMS ETL>
```

rmse_aip_config.env Settings for IP

There are variables you must change depending upon your local settings:

For example:

```
export DBNAME=int9i
export RMS_OWNER=steffej_rms1011
export BA_OWNER=rmsint1011
```

You must set up the environment variable PASSWORD in the rmse_aip_config.env or some other location that can be referenced. In the example below, adding the line to the rmse_aip_config.env causes the password 'mypasswd' to be used to log into the database:

```
export PASSWORD=mypasswd
```

Make sure to review the environmental parameters in the rmse_aip_config.env file file before executing batch modules.

Steps to Configure RETL

1. Log in to the UNIX server with a UNIX account that will run the RETL scripts.
2. Change directories to `<base_directory>/rfx/etc`.
3. Modify the rmse_aip_config.env script.

For example:

a. Change the DBNAME variable to the name of the RMS database.
b. Change the RMS_OWNER variable to the username of the RMS schema owner.
c. Change the BA_OWNER variable to the username of the RMSE batch user.

Program Return Code

RETL programs use one return code to indicate successful completion. If the program successfully runs, a zero (0) is returned. If the program fails, a non-zero is returned.

Program Status Control Files

To prevent a program from running while the same program is already running against the same set of data, the code utilizes a program status control file. At the beginning of each module, rmse_aip_config.env is run. These files check for the existence of the program status control file. If the file exists, then a message stating, '${PROGRAM_NAME} has already started', is logged and the module exits. If the file does not exist, a program status control file is created and the module executes.

If the module fails at any point, the program status control file is not removed, and the user is responsible for removing the control file before re-running the module.

File Naming Conventions

The naming convention of the program status control file allows a program whose input is a text file to be run multiple times at the same time against different files.

The name and directory of the program status control file is set in the applicable configuration file (rmse_aip_config.env). The directory defaults to $MMHOME/error. The naming convention for the program status control file itself defaults to the following dot separated file name:

- The program name
- 'status'
- The business virtual date for which the module was run

For example, a program status control file for one program would be named as follows for the batch run of January 5, 2001:

```
$MMHOME/error/rmse_aip_banded_item.status.20010105
```

Restart and Recovery

Because RETL processes all records as a set, as opposed to one record at a time, the method for restart and recovery must be different from the method that is used for Pro*C. The restart and recovery process serves the following two purposes:

1. It prevents the loss of data due to program or database failure.
2. It increases performance when restarting after a program or database failure by limiting the amount of reprocessing that needs to occur.

The RMS Extract (RMSE) modules extract from a source transaction database or text file and write to a text file. The RMS Load (RMSL) modules import data from flat files, perform transformations if necessary and then load the data into the applicable RMS tables.

Most modules use a single RETL flow and do not require the use of restart and recovery. If the extraction process fails for any reason, the problem can be fixed, and the entire process can be run from the beginning without the loss of data. For a module that takes a text file as its input, the following two choices are available that enable the module to be re-run from the beginning:

1. Re-run the module with the entire input file.
2. Re-run the module with only the records that were not processed successfully the first time and concatenate the resulting file with the output file from the first time.

To limit the amount of data that needs to be re-processed, more complicated modules that require the use of multiple RETL flows utilize a bookmark method for restart and recovery. This method allows the module to be restarted at the point of last success and run to completion. The bookmark restart/recovery method incorporates the use of a bookmark flag to indicate which step of the process should be run next. For each step in the process, the bookmark flag is written to and read from a bookmark file.

Note: If the fix for the problem causing the failure requires changing data in the source table or file, then the bookmark file must be removed and the process must be re-run from the beginning in order to extract the changed data.

Message Logging

Message logs are written daily in a format described in this section.

Daily Log File

Every RETL program writes a message to the daily log file when it starts and when it finishes. The name and directory of the daily log file is set in the configuration file (rmse_aip_config.env). The directory defaults to $MMHOME/log. All log files are encoded UTF-8.

The naming convention of the daily log file defaults to the following "dot" separated file name:

- The business virtual date for which the modules are run
- '.log'

For example, the location and the name of the log file for the business virtual date of January 5, 2001 would be the following:

```
$MMHOME/log/20010105.log
```

Format

As the following examples illustrate, every message written to a log file has the name of the program, a timestamp, and either an informational or error message:

```
aipt_item 17:07:43: Program started ...
aipt_item 17:07:50: Program completed successfully
rmse_aip_item_master 17:08:53: Program started ...
rmse_aip_item_master 17:08:59: Program completed successfully
rmse_item_retail 17:09:07: Program started ...
rmse_item_retail 17:09:12: Program completed successfully
```

If a program finishes unsuccessfully, an error file is usually written that indicates where the problem occurred in the process. There are some error messages written to the log file, such as 'No output file specified', that require no further explanation written to the error file.

Program Error File

In addition to the daily log file, each program also writes its own detail flow and error messages. Rather than clutter the daily log file with these messages, each program writes out its errors to a separate error file unique to each execution.

The name and directory of the program error file is set in the applicable configuration file (rmse_aip_config.env). The directory defaults to $MMHOME/error. All errors and *all routine processing messages* for a given program on a given day go into this error file (for example, it will contain both the stderr and stdout from the call to RETL). All error files are encoded UTF-8.

The naming convention for the program's error file defaults to the following "dot" separated file name:

- The program name
- The business virtual date for which the module was run

For example, all errors and detail log information for the `rms_aip_item_master` program would be placed in the following file for the batch run of January 5, 2001:

```
$MMHOME/error/rms_aip_item_master.20010105
```

RMSE and Transformation Reject Files

RMSE extract and transformation modules may produce a reject file if they encounter data related problems, such as the inability to find data on required lookup tables. The module tries to process all data and then indicates that records were rejected so that all data problems can be identified in one pass and corrected; then, the module can be re-run to successful completion. If a module does reject records, the reject file is *not* removed, and the user is responsible for removing the reject file before re-running the module.

The records in the reject file contain an error message and key information from the rejected record. The following example illustrates a record that is rejected due to problems within the currency conversion library:

```
Currency Conversion Failed|101721472|20010309
```

The following example illustrates a record that is rejected due to problems looking up information on a source table:

```
Unable to find item_master record for Item|101721472
```

The name and directory of the reject file is set in the applicable configuration file (rmse_config.env or config.env). The directory defaults to $MMHOME/data.

Note: A directory specific to reject files can be created. The rmse_config.env and/or config.env file would need to be changed to point to that directory.

The naming convention for the reject file defaults to the following "dot" separated file name:

- The program name
- The first filename, if one is specified on the command line
- 'rej'
- The business virtual date for which the module was run

For example, all rejected records for the `slsildmex` program would be placed in the following file for the batch run of January 5, 2001:

`$MMHOME/data/slsildmex.slsildmdm.txt.rej.20010105`

Schema Files Overview

RETL uses schema files to specify the format of incoming or outgoing datasets. The schema file defines each column's data type and format, which is then used within RETL to format/handle the data. For more information about schema files, see the latest RETL Programmer's Guide. Schema file names are hard-coded within each module since they do not change on a day-to-day basis. All schema files end with ".schema" and are placed in the "rfx/schema" directory.

Command Line Parameters

In order for each RETL module to run, the input/output data file paths and names may need to be passed in at the UNIX command line.

RMSE and Transformation

Most RMSE and transformation modules do not require the passing in of any parameters. The output path/filename defaults to $DATA_DIR/(RMSE and transfer program name).dat. Similarly, the schema format for the records in these files are specified in the file - $SCHEMA_DIR/(RMSE program name).schema

Scripts that need Parameter to Run

The scripts below are run on a full snapshot basis. The parameter is `F` (for full snapshot).

- rmse_aip_store_cur_inventory.ksh
- rmse_aip_wh_cur_inventory.ksh

Typical Run and Debugging Situations

The following examples illustrate typical run and debugging situations for types of programs. The log, error, and so on file names referenced below assume that the module is run on the business virtual date of March 9, 2001. See the previously described naming conventions for the location of each file.

For example:

To run rmse_aip_store.ksh:

1. Change directories to $MMHOME/rfx/src.
2. At a UNIX prompt enter:

```
%rmse_aip_store.ksh
```

If the module runs successfully, the following results:

1. **Log file:** Today's log file, 20010309.log, contains the messages "Program started …" and "Program completed successfully" for rmse_aip_store.
2. **Data:** The rmse_aip_store.dat file exists in the data directory and contains the extracted records.
3. **Schema:** The rmse_aip_store.schema file exists in the schema directory and contains the definition of the data file in #2 above.
4. **Error file:** The program's error file, rmse_aip_store.20010309, contains the standard RETL flow (ending with "All threads complete" and "Flow ran successfully") and no additional error messages.
5. **Program status control:** The program status control file, rmse_aip_store.status.20010309, does not exist.
6. **Reject file:** The reject file, rmse_aip_store.rej.20010309, does not exist because this module does not reject records.

If the module does *not* run successfully, the following results:

1. **Log file:** Today's log file, 20010309.log, does not contain the "Program completed successfully" message for rmse_stores.
2. **Data:** The rmse_aip_store.dat file may exist in the data directory but may not contain all the extracted records.
3. **Schema:** The rmse_aip_store.schema file exists in the schema directory and contains the definition of the data file in #2 above.
4. **Error file:** The program's error file, rmse_aip_store.20010309, may contain an error message.
5. **Program status control:** The program status control file, rmse_aip_store.status.20010309, exists.
6. **Reject file:** The reject file, rmse_aip_store.status.20010309, does not exist because this module does not reject records.

To re-run the module, perform the following actions:

1. Determine and fix the problem causing the error.
2. Remove the program's status control file.
3. Change directories to $MMHOME/rfx/src. At a UNIX prompt, enter:

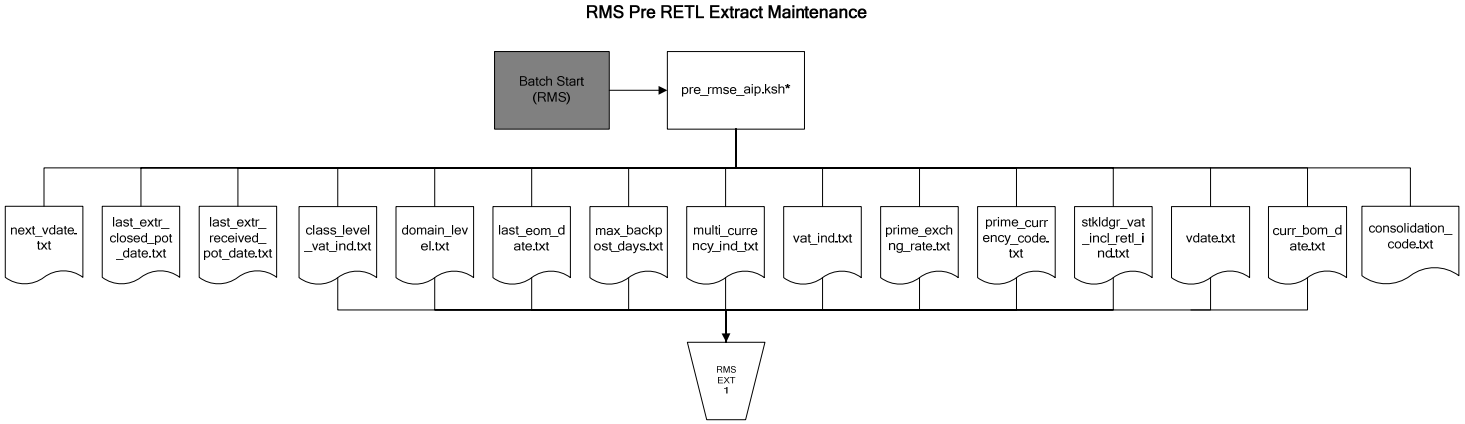
```
%rmse_aip_store.ksh
```

Program Flow Diagrams

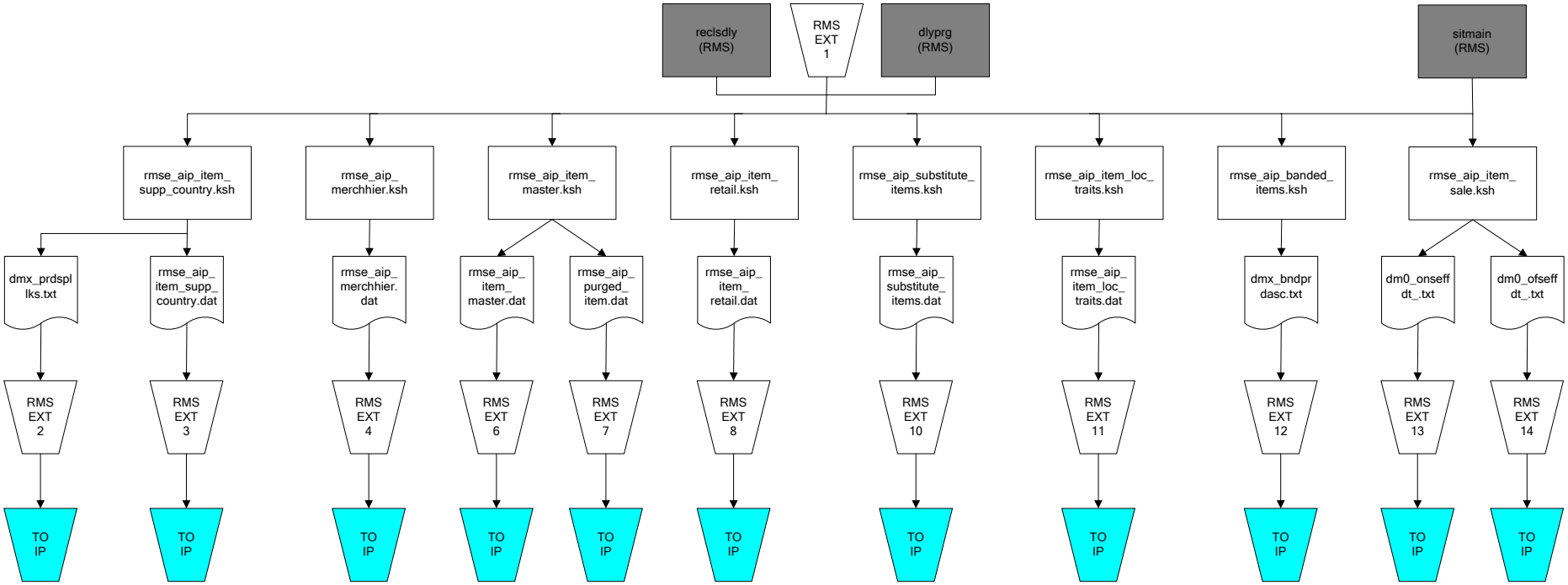
This section presents flow diagrams for data processing from sources. The source system's program or output file is illustrated along with the program or process that interfaces with the source. After initial interface processing of the source, the diagrams illustrate the flow of the data.

Before setting up a program schedule, familiarize yourself with the functional and technical constraints associated with each program.

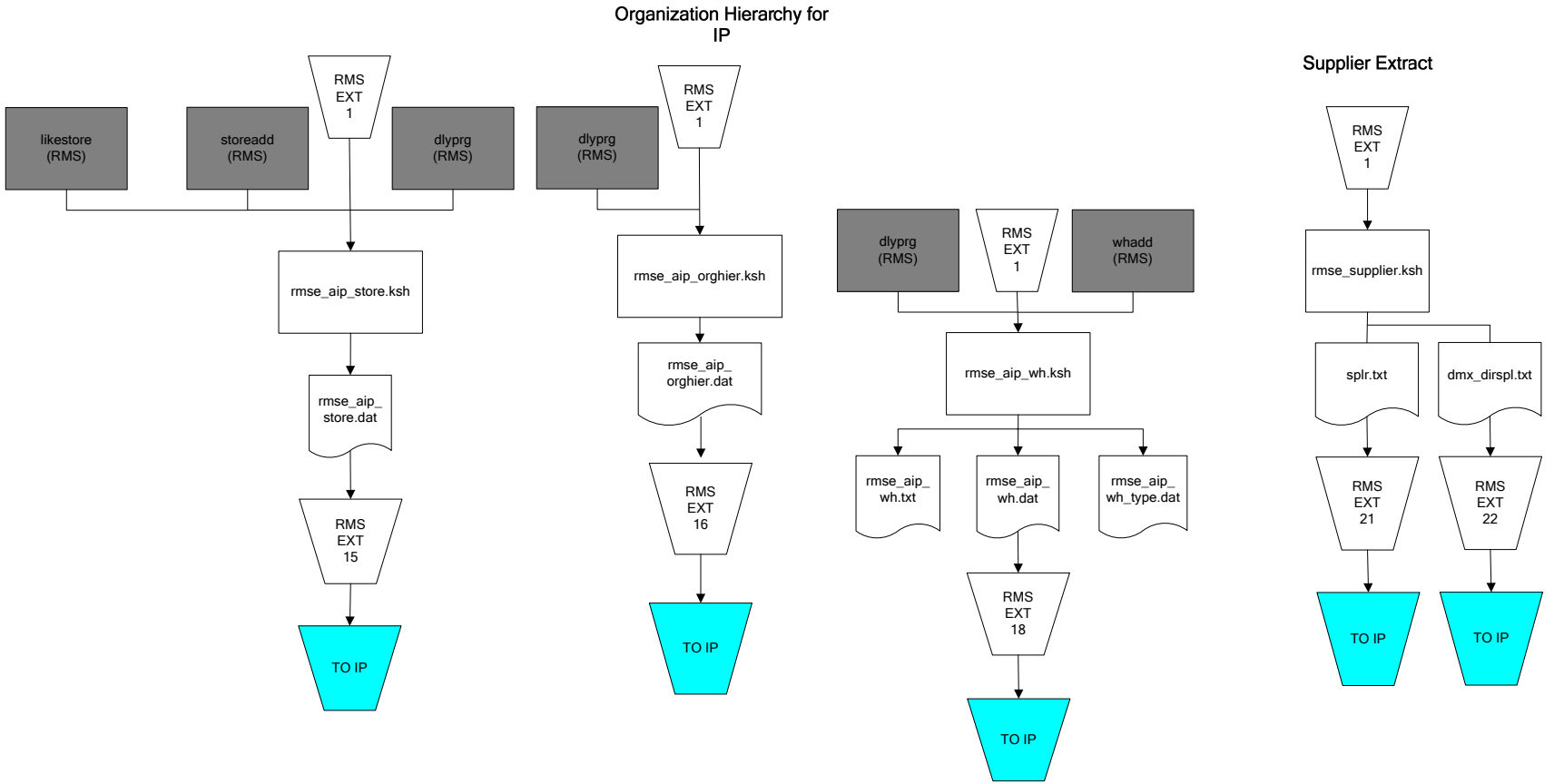
RMS Pre/Post Extract Diagrams



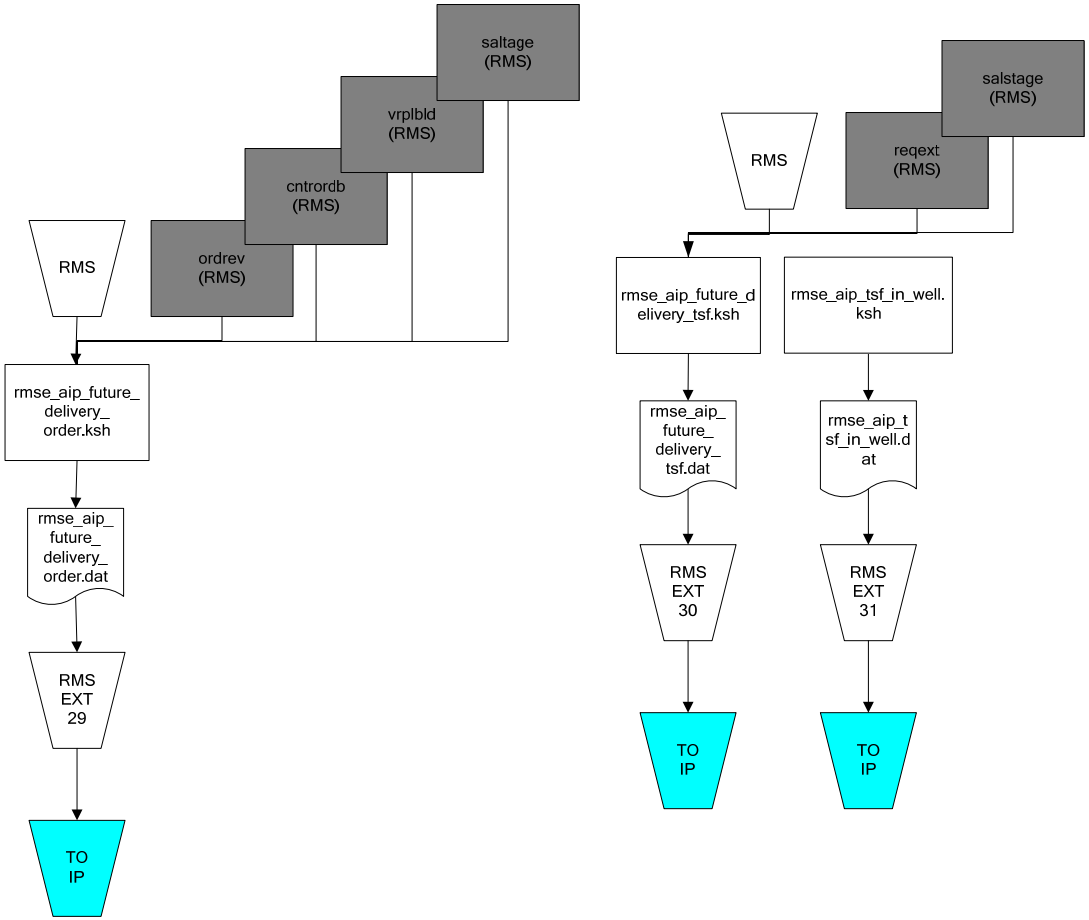
RMS Foundation Data Extract Diagrams



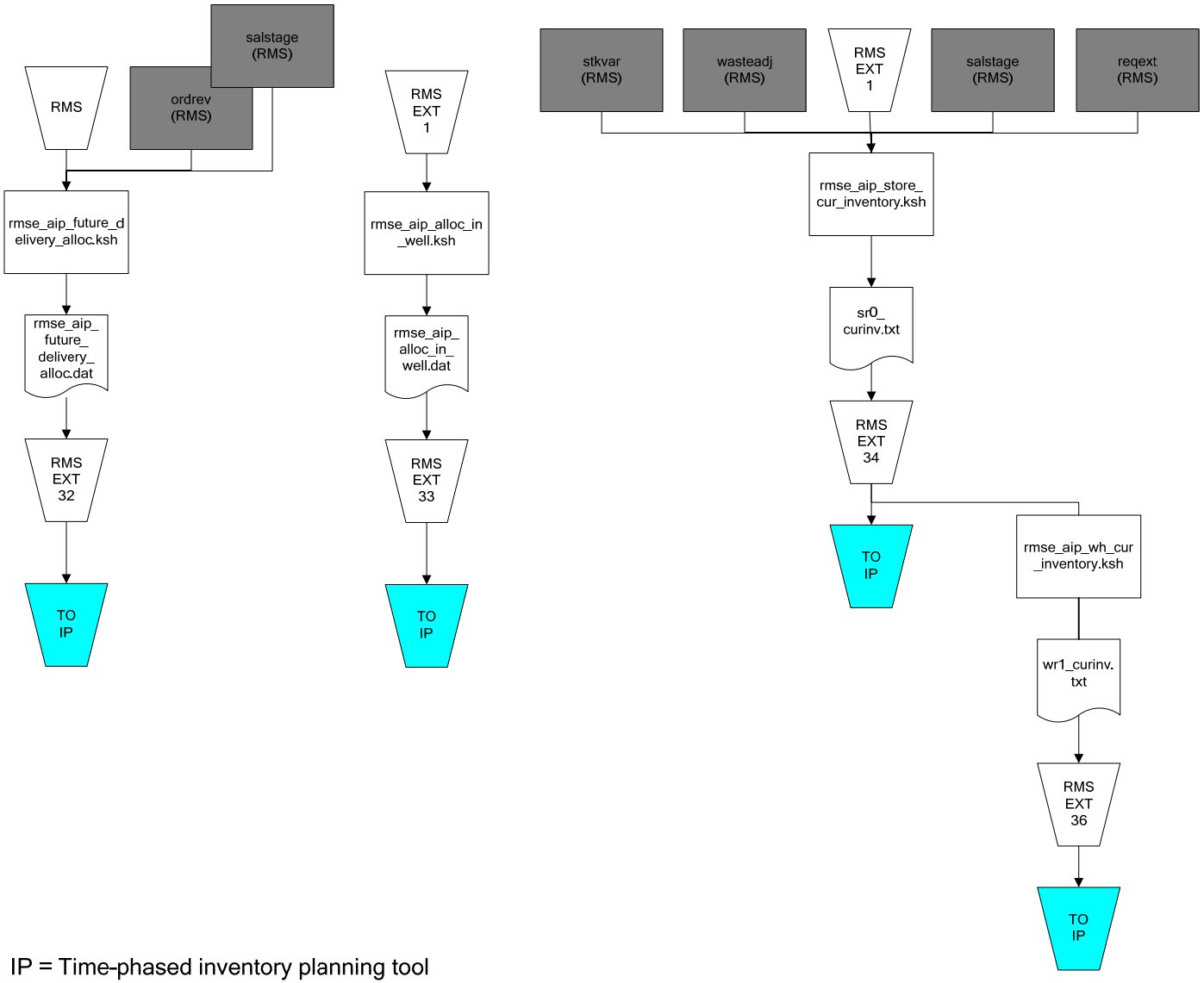
IP = Time-phased inventory planning tool



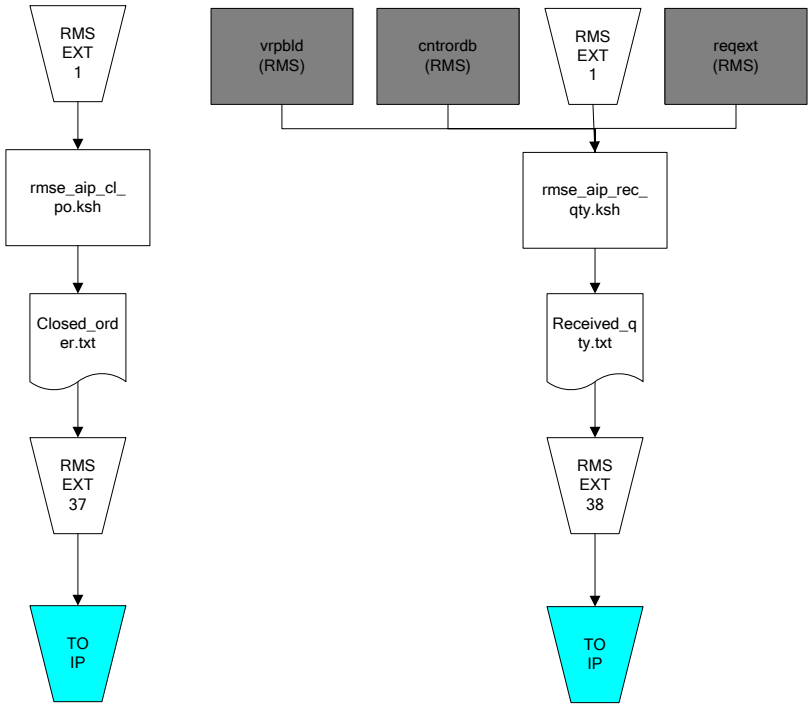
IP = Time-phased inventory planning tool



IP = Time-phased inventory planning tool



IP = Time-phased inventory planning tool



IP = Time-phased inventory planning tool

Naming Conventions

Notes on the columns in the following RETL extraction programs table:

- The "Extraction Program Name" column includes the full name of the extract script. The results of these scripts are stored in "rmse_rpas_<basename>.dat", and the schemas are specified in "rmse_rpas_<basename>.schema".
- The "Column extracted" column refers to the column name in the source database table.
- The "Column type" column refers to the datatype in the source database table.
- The "Target field" column refers to the name of the field as specified in the schema file for the related extract.
- The "Field type and length "column refers to the datatype of the field as specified in the schema file for the related extract.

RETL Programs that Extract from RMS

rmse_aip_alloc_in_well (RMS Extract of Allocations in the Well Quantities to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_alloc_in_well.ksh

Design Overview

This script extracts RMS "in the well" allocation quantities for integration with a time-phased inventory planning tool. In the well pertains to inventory that has been reserved by allocations in approved or reserved status. The expected release date is also included in the extract.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After pre_rmse_aip.ksh
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ITEM_MASTER	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
ITEM_SUPPLIER	Yes	No	No	No
ORDHEAD	Yes	No	No	No
ALLOC_HEADER	Yes	No	No	No
ALLOC_DETAIL	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
PACKITEM	Yes	No	No	No

I/O Specification

Output File Layout

The output file rmse_aip_alloc_in_well.dat is in fixed-length format matching the schema definition in rmse_aip_alloc_in_well.schema.

Field Name	Field Type	Required	Description
DAY	Char(9)	Yes	Current date if alloc_header.release_date is less than current date else alloc_header.release_date
LOC	Integer(20)	Yes	Alloc_header.wh
ITEM	Char(20)	Yes	Formal Case Type: If simple pack then and alloc_detail.to_loc_type = 'S' then this would be the component of the pack in v_packsku_qty else item_master.item. Informal Case Type: Item_master.item

Field Name	Field Type	Required	Description
ORDER_MULTIPLE	Integer(4)	Yes	Formal Case Type: If simple pack and alloc_detail.to_loc_type = 'W' then this would be v_packsku_qty.qty of the pack component else 1 Informal Case Type: One unique record for each item/supplier with order multiples of: 1, supp_pack_size, inner_pack_size and (ti * hi * supp_packsize)
ALLOC_RESERVE_QTY	Integer(8)	Yes	Formal Case Type: Alloc_detail.qty_allocated – alloc_detail.qty_received. Resulting quantity is multiplied by V_packsku_qty.qty if item is a pack. Informal Case Type: Alloc_detail.qty_allocated – alloc_detail.qty_received expressed in multiples of the primary case size. The remainder is expressed in Standard UOM.

rmse_aip_banded_item (RMS Extract of Banded Item Information to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_banded_item.ksh

Design Overview

This script extracts RMS banded items and their associated "promotional item" or substitute item.

The association between the banded item (component) and its promotional item (substitute item) is established by joining item_master, sub_items_detail and v_packsku_qty for formal pack items, and item_master, sub_items_detail and item_supp_country for informal pack items. Items that have a banded item ind = 'Y' are joined with sub_items_detail on item. The associated promotional item would be the sub_item. For formal pack items, v_pack_sku.qty gives the order_multiple for both the standard item and its promotional item. Form informal pack items, the different pack sizes (inner, case, pallet) are obtained from item_supp_country for both the standard and promotional item. The standard item's time-phased inventory planning tool case type decides whether we get the pack sizes for both standard and promotional items from v_packsku_qty or item_supp_country.

Additional conditions on the extract are as follows:

- Both banded and promotional item are in approved status.
- Both banded and promotional item should be forecastable (item_master.forecast_ind = 'Y').
- In case of informal pack items, the pack size extracted for both banded and promotional item is for the primary supplier and primary supplier country.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After pre_rmse_aip.ksh.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ITEM_MASTER	Yes	No	No	No
SUB_ITEMS_DETAIL	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No

I/O Specification

Output File Layout

The dmx_bndprdasc.txt is in fixed-length format matching the schema definition in rmse_aip_dmx_bndprdasc.schema.

Field Name	Field Type	Required	Description
STANDARD_SKU	Char(20)	Yes	Item_master.item
STANDARD_ORDER_ MULTIPLE	Integer(4)	Yes	For informal pack items: 1, Item_supp_country.inner_pack_size, Item_supp_country.supp_pack_size, Item_supp_country.supp_pack_size * hi * ti For formal pack items: V_packsku_qty.qty, 1
PROMOTIONAL_SKU	Char(20)	Yes	Sub_items_detail.sub_item
PROMOTIONAL_ORDER_MU LTIPLE	Integer(4)	Yes	For informal pack items: 1, Item_supp_country.inner_pack_size, Item_supp_country.supp_pack_size, Item_supp_country.supp_pack_size * hi * ti For formal pack items: V_packsku_qty.qty, 1

rmse_aip_cl_po (RMS Extract of Cancelled or Closed IP POs and Transfers to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_cl_po.ksh

Design Overview

This script extracts from RMS cancelled or closed purchase orders and transfers for integration with a time-phased inventory planning tool. Only records that meet the following criteria below are extracted:

For Purchase Orders:

- Ordhead.close_date is not NULL
- Ordhead.orig_ind = 6 (external system generated)
- Ordhead.close_date > Retl_extract_dates.last_extr_closed_pot_date

For Transfers:

- Tsfhead.close_date is not NULL
- Tsfhead.tsf_type = 'AIP' (generated by the time-phased inventory planning tool)
- Ordhead.close_date > Retl_extract_dates.last_extr_closed_pot_date

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	Before tsfprg.pc and ordprg.pc. After pre_rmse_aip.ksh.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ORDHEAD	Yes	No	No	No
TSFHEAD	Yes	No	No	No

Rmse_aip_cl_po.ksh calls another script rmsl_aip_update_retl_date.ksh, which updates the IP RETL extract dates. The table affected by this script is:

Table	Select	Insert	Update	Delete
RETL_EXTRACT_DATES	No	No	Yes	No

I/O Specification

Output File Layout

The output file closed_order.txt is in fixed-length format matching the schema definition in rmse_aip_cl_po.schema.

Field Name	Field Type	Required	Description
ORDER_NUMBER	Integer(10)	Yes	Ordhead.order_no or tsfhead.tsf_no
ORDER_TYPE	Char(1)	Yes	'P' for purchase orders or 'T' for transfers

rmse_aip_future_delivery_alloc (RMS Extract of Allocation Quantities for Future Delivery to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_future_delivery_alloc.ksh

Design Overview

This script extracts RMS in-transit and on-order allocation quantities for future delivery for integration with a time-phased inventory planning tool.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After pre_rmse_aip.ksh
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ITEM_MASTER	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
ITEM_SUPPLIER	Yes	No	No	No
ORDHEAD	Yes	No	No	No
ALLOC_HEADER	Yes	No	No	No
ALLOC_DETAIL	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No

Table	Select	Insert	Update	Delete
PACKITEM	Yes	No	No	No
TRANSIT_TIMES	Yes	No	No	No
V_WH	Yes	No	No	No

I/O Specification

Output File Layout

The output file rmse_aip_future_delivery_alloc.dat is in fixed-length format matching the schema definition in rmse_aip_future_delivery_alloc.schema.

Field Name	Field Type	Required	Description
DAY	Char(9)	Yes	'D' || Current date if Alloc_header.release_date + transit_times.transit_time is less than current date else 'D' || Alloc_header.release_date + transit_times.transit_time
SUPPLIER	Integer(20)	No	If there is no associated order then primary supplier on item_supplier.supplier else ordhead.supplier
LOC	Integer(20)	Yes	Alloc_detail.to_loc
ITEM	Char(20)	Yes	Formal Case Type: If simple pack then and alloc_detail.to_loc_type = 'S' then this would be the component of the pack in v_packsku_qty else item_master.item. Informal Case Type: Item_master.item
ORDER_MULTIPLE	Integer(4)	Yes	Formal Case Type: V_packsku_qty.qty for simple pack, else 1 Informal Case Type: One unique record for each item/supplier with order multiples of: 1, supp_pack_size, inner_pack_size and (ti * hi * supp_packsize)
IN_TRANSIT_ALLOC_QTY	Integer(8)	Yes	Formal Case Type: Alloc_detail.Qty_transferred – Alloc_detail.Qty_received. Resulting quantity is multiplied by V_packsku_qty.qty if item is a pack. Informal Case Type: Alloc_detail.Qty_transferred – Alloc_detail.Qty_received expressed in the primary case size. Remainder is in Standard UOM

Field Name	Field Type	Required	Description
ON_ORDER_ALLOC_QTY	Integer(8)	Yes	Formal Case Type: Alloc_detail.Qty_allocated – Alloc_detail.Qty_transferred. Resulting quantity is multiplied by V_packsku_qty.qty if item is a pack. Informal Case Type: Alloc_detail.Qty_allocated – Alloc_detail.Qty_transferred expressed in the primary case size. Remainder is in Standard UOM

rmse_aip_future_delivery_order (RMS Extract of Purchase Order Quantities for Future Delivery to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_future_delivery_order.ksh

Design Overview

This script extracts RMS purchase order quantities for future delivery for integration with a time-phased inventory planning tool.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After vrplbld.pc, cntrordb.pc and pre_rmse_aip.ksh.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ITEM_SUPP_COUNTRY	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
ITEM_SUPPLIER	Yes	No	No	No
ORDHEAD	Yes	No	No	No
ORDLOC	Yes	No	No	No
ALLOC_HEADER	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
PACKITEM	Yes	No	No	No

I/O Specification

Output File Layout

The item output file is in fixed-length format matching to the schema definition in rmse_aip_future_delivery_order.schema.

Field Name	Field Type	Required	Description
DAY	Char(9)	Yes	'D' || Period.vdate if ordhead.not_after_date < period.vdate else 'D' || Ordhead.not_after_date
SUPPLIER	Integer(20)	Yes	Ordhead.supplier
LOC	Integer(20)	Yes	Ordloc.location
ITEM	Char(20)	Yes	Formal Case Type: If simple pack and ordloc.loc_type = 'S' then this would be the component of the pack in v_packsku_qty else item_master.item. Informal Case Type: Item_master.item

Field Name	Field Type	Required	Description
ORDER_MULTIPLE	Integer(4)	Yes	Formal Case Type: If ordloc.loc_type = 'S' then 1 If ordloc.loc_type = 'W' and (ordloc.qty_ordered – ordloc.qty_received) >= item_supp_country.supp_pack_size and a simple pack then V_packsku_qty.qty else 1 Informal Case Type: One unique record for each item/supplier with order multiples of: 1, supp_pack_size, inner_pack_size and (ti * hi * supp_packsize)
PO_QTY	Integer(8)	Yes	(Ordloc.qty_ordered – Ordloc.qty_received) or 0
CUST_ORDER	Char(1)	Yes	Ordhead.cust_order
LOC_TYPE	Char(1)	Yes	Ordloc.loc_type

rmse_aip_future_delivery_tsf (RMS Extract of On-order and In-transit Transfer Quanties for Future Delivery to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_future_delivery_tsf.ksh

Design Overview

This script extracts RMS on-order and in-transit transfer quantities for future delivery for integration with a time-phased inventory planning tool.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After reqext.pc and pre_rmse_aip.ksh
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ITEM_MASTER	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
ITEM_SUPPLIER	Yes	No	No	No
TSFHEAD	Yes	No	No	No
TSFDETAIL	Yes	No	No	No
SHIPITEM_INV_FLOW	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
PACKITEM	Yes	No	No	No
TRANSIT_TIMES	Yes	No	No	No
V_WH	Yes	No	No	No

I/O Specification

Output File Layout

The output file rmse_aip_future_delivery_tsf.dat is in fixed-length format matching the schema definition in rmse_aip_future_delivery_tsf.schema.

Field Name	Field Type	Required	Description
DAY	Char(9)	Yes	'D' || current date if tsfhead.delivery_date + transit_times.transit_time is less than current date else 'D' || tsfhead.delivery_date + transit_times.transit_time
SUPPLIER	Integer(20)	No	Item_supp_country.supplier
LOC	Integer(20)	Yes	Shipitem_inv_flow.to_loc if tsfhead.to_loc_type = 'W' and tsfhead.tsf_type = 'EG else Tsfhead.to_loc

Field Name	Field Type	Required	Description
ITEM	Char(20)	Yes	Formal Case Type: If simple pack and tsfhead.to_loc_type = 'S' then this would be the component of the pack in v_packsku_qty else item_master.item. Informal Case Type: Item_master.item
ORDER_MULTIPLE	Integer(4)	Yes	Formal Case Type: If simple pack and tsfhead.to_loc_type = 'W' the v_packsku_qty.qty else 1 Informal Case Type: One unique record for each item/supplier with order multiples of: 1, supp_pack_size, inner_pack_size and (ti * hi * supp_packsize)
TSF_QTY	Integer(8)	Yes	Formal Case Type: Tsfdetail.tsf_qty – tsfdetail.received_qty. Resulting quantity is multiplied by V_packsku_qty.qty if item is a pack. Informal Case Type: Tsfdetail.tsf_qty – tsfdetail.received_qty expressed in the primary case size. Remainder is in Standard UOM
IN_TRANSIT_TSF_QTY	Integer(8)	Yes	Formal Case Type: Tsfdetail.ship_qty – tsfdetail.received_qty. Resulting quantity is multiplied by V_packsku_qty.qty if item is a pack. Informal Case Type: Tsfdetail.ship_qty – tsfdetail.received_qty expressed in the primary case size. Remainder is in Standard UOM
ON_ORDER_TSF_QTY	Integer(8)	Yes	Formal Case Type: Tsfdetail.tsf_qty – tsfdetail.ship_qty. Resulting quantity is multiplied by V_packsku_qty.qty if item is a pack. Informal Case Type: Tsfdetail.tsf_qty – tsfdetail.ship_qty expressed in the primary case size. Remainder is in Standard UOM.
LOC_TYPE	Char(1)	Yes	Tsfhead.to_loc_type
TSF_TYPE	Char(6)	Yes	Tsfhead.tsf_type

rmse_aip_item_loc_traits (RMS Extract of Item Location Traits to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_item_loc_traits.ksh

Design Overview

This script extracts from RMS item location traits information for integration with a time-phased inventory planning tool. Only the following items are extracted:

- Approved, non-pack and forecastable
- Approved and a simple pack item whose component is forecastable.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After pre_rmse_aip.ksh.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ITEM_LOC_TRAITS	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No

I/O Specification

Output File Layout

The output file rmse_aip_item_loc_traits.dat is in fixed-length format matching the schema definition in rmse_aip_item_loc_traits.schema.

Field Name	Field Type	Required	Description
ITEM	Char(25)	Yes	Item_master.item
LOC	Integer(10)	Yes	Item_loc_traits.loc
REQ_SHELF_LIFE_ ON_RECEIPT	Integer(8)	No	Item_loc_traits.req_shelf_life_on_ receipt

rmse_aip_item_master (RMS Extract of Items to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_item_master.ksh

Design Overview

This script extracts RMS item information for integration with a time-phased inventory planning tool.

Two output files are produced by this extract. One contains approved transaction-level items while the other contains purged items from the daily_purge table.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After sitmain.pc, reclsdly.pc and pre_rmse_aip.ksh. Before dlyprg.pc
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ITEM_MASTER	Yes	No	No	No
ITEM_SUPPLIER	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
UOM_CLASS	Yes	No	No	No
CODE_DETAIL	Yes	No	No	No
DAILY_PURGE	Yes	No	No	No

I/O Specification

Output File Layout

The item output file is in fixed-length format matching to the schema definition in rmse_aip_item_master.schema.

Field Name	Field Type	Required	Description
ITEM	Char(25)	Yes	Item_master.item
ITEM_DESC	Char(100)	Yes	Item_master.item_desc
RMS_SKU_DESCRIPTION	Char(60)	Yes	First 60 characters of Item_master.item_desc
ITEM_PARENT	Char(25)	No	Item_master.item_parent
ITEM_GRANDPARENT	Char(25)	No	Item_master.item_grandparent
AIP_SKU	Char(25)	Yes	V_packsku_qty.item or Item_master.item
SUBCLASS	Integer(5)	Yes	Item_master.subclass
CLASS	Integer(5)	Yes	Item_master.class
DEPT	Integer(5)	Yes	Item_master.dept
FORECAST_IND	Char(1)	Yes	Item_master.forecast_ind
SUPPLIER	Integer(11)	Yes	Item_supplier.supplier
PRIMARY_SUPP_IND	Char(1)	Yes	Item_supplier.primary_supp_ind
STANDARD_UOM	Char(4)	Yes	Item_master.standard_uom
STANDARD_UOM_DESCRIPTION	Char(20)	Yes	Uom_class.uom_desc
SKU_TYPE	Char(6)	No	Item_master.handling_temp or 0
SKU_TYPE_DESCRIPTION	Char(40)	No	Code_detail.code_desc (for code_type 'HTMP')
PACK_QUANTITY	Integer(4)	No	V_packsku_qty.qty or 0

Field Name	Field Type	Required	Description
PACK_IND	Char(1)	Yes	Item_master.pack_ind
SIMPLE_PACK_IND	Char(1)	Yes	Item_master.simple_pack_ind
ITEM_LEVEL	Integer(1)	Yes	Item_master.item_level
TRAN_LEVEL	Integer(1)	Yes	Item_master.tran_level
RETAIL_LABEL_TYPE	Char(6)	No	Item_master.retail_label_type
BANDED_ITEM_IND	Char(1)	No	1 if Item_master.banded_item_ind = 'Y' and 0 if Item_master.banded_item_ind = 'N'
CATCH_WEIGHT_IND	Char(1)	Yes	Item_master.catch_weight_ind
SELLABLE_IND	Char(1)	Yes	Item_master.sellable_ind
ORDERABLE_IND	Char(1)	Yes	Item_master.orderable_ind
DEPOSIT_ITEM_TYPE	Char(6)	No	Item_master.deposit_item_type

The purged items output file is in fixed-length format matching to the schema definition in rmse_aip_purged_item.schema.

Field Name	Field Type	Required	Description
ITEM	Char(25)	Yes	Daily_purge.key_value

rmse_aip_item_retail (RMS Extract of Item Retail to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_item_retail.ksh

Design Overview

This script extracts from RMS item information required by the item transformation script aipt_item.ksh for integration with a time-phased inventory planning tool. Records that meet the following criteria are extracted:

Non-pack items

- Approved and transaction level items
- Have supplier pack sizes greater than 1
- Forecastable (item_master.forecast_ind = 'Y')
- Inventory items

Simple pack components

- Component of approved and transaction level simple packs
- Components are forecastable (item_master.forecast_ind = 'Y')
- Simple packs are inventory items

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After pre_rmse_aip.ksh.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ITEM_MASTER	Yes	No	No	No
ITEM_SUPPLIER	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
UOM_CLASS	Yes	No	No	No
CODE_DETAIL	Yes	No	No	No

I/O Specification

Output File Layout

The output file rmse_aip_item_retail.dat is in fixed-length format matching the schema definition in rmse_aip_item_retail.schema.

Field Name	Field Type	Required	Description
ITEM	Char(25)	Yes	Item_master.item
RMS_SKU_DESCRIPTION	Char(60)	Yes	First 60 characters of item_master.item_desc
AIP_SKU	Char(25)	Yes	Item_master.item
SUBCLASS	Integer(5)	Yes	Item_master.subclass
CLASS	Integer(5)	Yes	Item_master.class
DEPT	Integer(5)	Yes	Item_master.dept

Field Name	Field Type	Required	Description
STANDARD_UOM	Char(4)	Yes	Item_master.standard_uom
STANDARD_UOM_ DESCRIPTION	Char(20)	Yes	Uom_class.uom_desc_standard
SKU_TYPE	Char(6)	No	Non-pack items Item_master.handling_temp. "0" if NULL. Simple pack components Item_master.handling_temp or NULL.
SKU_TYPE_ DESCRIPTION	Char(40)	No	Non-pack items Code_detail.code_desc . "0" if NULL. Simple pack components Code_detail.code_desc or NULL.
ORDER_MULTIPLE	Integer(4)	Yes	1
PACK_QUANTITY	Integer(4)	No	0
BANDED_ITEM_IND	Char(1)	No	"1" if item_master.banded_item_ind = "Y' else "0"

rmse_aip_item_sale (RMS Extract of On/Off Sale to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_item_sale.ksh

Design Overview

This script extracts from RMS on/off sale information for integration with a time-phased inventory planning tool. This information contains the status, status update date and order multiple for an item/location. A status of 'A' indicates that an item/location is valid and can be ordered and sold. A status of 'C' indicates than an item/location is invalid and cannot be ordered or sold. The script only extracts items that meet the following criteria:

- In active status
- Transaction-level
- Either non-pack or a simple pack
- Sit_detail.status is either 'A' or 'C'
- Sit_detail.status_update_date is greater than the current date

Only the order multiple for the primary supplier and primary supplier country is extracted.

The script produces two output files, one containing on sale records (sit_detail.status = 'A') and the other off sale records (sit_detail.status = 'C').

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After sitmain.pc and pre_rmse_aip.ksh.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ITEM_MASTER	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
SIT_EXPLODE	Yes	No	No	No
SIT_DETAIL	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No

I/O Specification

Output File Layout

The output file dm0_onseffdt_.txt is in fixed-length format matching the schema definition in rmse_aip_item_on_sale.schema.

Field Name	Field Type	Required	Description
STORE	Integer(20)	Yes	Sit_explode.location
RMS_SKU	Char(20)	Yes	Sit_explode.item

Field Name	Field Type	Required	Description
ORDER_MULTIPLE	Integer(4)	Yes	If item_master.pack_ind = 'Y' then v_packsku_qty.qty (for the component item) else item_supp_country.order_multiple
ON_SALE_EFFECTIVE_DATE	Date	Yes	Sit_detail.status_update_date

The output file dm0_ofseffdt_.txt is in fixed-length format matching the schema definition in rmse_aip_item_off_sale.schema.

Field Name	Field Type	Required	Description
STORE	Integer(20)	Yes	Sit_explode.location
RMS_SKU	Char(20)	Yes	Sit_explode.item
ORDER_MULTIPLE	Integer(4)	Yes	If item_master.pack_ind = 'Y' then v_packsku_qty.qty (for the component item) else item_supp_country.order_multiple
OFF_SALE_EFFECTIVE_DATE	Date	Yes	Sit_detail.status_update_date

rmse_aip_item_supp_country (RMS Extract of Item Supplier Country to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_item_supp_country.ksh

Design Overview

This script extracts RMS item-supplier information for integration with a time-phased inventory planning tool.

Three output files are produced by this extract. Two contain item-supplier information. The other is a reject file containing item suppliers with rejected order multiples.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After sitmain.pc, reclsdly.pc, pre_rmse_aip.ksh.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ITEM_MASTER	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
PACKITEM	Yes	No	No	No

I/O Specification

Output File Layout

The output file rmse_aip_item_supp_country.dat is in fixed-length format matching the schema definition in rmse_aip_item_supp_country.schema.

Field Name	Field Type	Required	Description
ITEM	Char(25)	Yes	Item_supp_country.item
SUPPLIER	Integer(11)	Yes	Item_supp_country.supplier
ORDER_MULTIPLE	Integer(4)	Yes	Formal Case Type: V_packsku_qty.qty for simple pack, else 1 Informal Case Type: One unique record for each item/supplier with order multiples of: 1, supp_pack_size, inner_pack_size and (ti * hi * supp_packsize)
PRIMARY_SUPP_IND	Char(1)	Yes	Item_supp_country.primary_supp_ind

The output file aip_dmx_prdsplks.txt is in fixed-length format matching the schema definition in rmse_aip_dmx_prdspllks.schema.

Field Name	Field Type	Required	Description
SUPPLIER	Integer(20)	Yes	Item_supp_country.supplier
RMS_SKU	Char(20)	Yes	Item_supp_country.item
ORDER_MULTIPLE	Integer(4)	Yes	Formal Case Type: V_packsku_qty.qty for simple pack, else 1 Informal Case Type: One unique record for each item/supplier with order multiples of: 1, supp_pack_size, inner_pack_size and (ti * hi * supp_packsize)
COMMODITY_SUPPLIER_LINKS	Char(1)	Yes	1

The reject file rmse_aip_item_supp_country_reject_ord_mult.txt is in pipe delimited (|) format.

Field Name	Field Type	Required	Description
ITEM	Char(25)	Yes	Item_supp_country.item
SUPPLIER	Integer(10)	Yes	Item_supp_country.supplier
ORDER_MULTIPLE	N/A (can exceed default limit of Integer (4) for order multiples	Yes	Formal Case Type: V_packsku_qty.qty for simple pack, else 1 Informal Case Type: One unique record for each item/supplier with order multiples of: 1, supp_pack_size, inner_pack_size and (ti * hi * supp_packsize)
PRIMARY_SUPP_IND	Char(1)	Yes	Item_supp_country.primary_supp_ind

rmse_aip_merchhier (RMS Extract of Merchandise Hierarchy to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_merchhier.ksh

Design Overview

This script extracts RMS merchandise hierarchy information for integration with a time-phased inventory planning tool.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After dlyprg.pc and pre_rmse_aip.ksh.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
SUBCLASS	Yes	No	No	No
CLASS	Yes	No	No	No
DEPS	Yes	No	No	No
GROUPS	Yes	No	No	No
DIVISION	Yes	No	No	No
COMPHEAD	Yes	No	No	No

I/O Specification

Output File Layout

The output file is in fixed-length format matching to the schema definition in rmse_aip_merchhier.schema.

Field Name	Field Type	Required	Description
SUBCLASS	Integer(5)	Yes	Subclass.subclass
SUB_NAME	Char(20)	Yes	Subclass.sub_name
CLASS	Integer(5)	Yes	Subclass.class
CLASS_NAME	Char(20)	Yes	Class.class_name
DEPT	Integer(5)	Yes	Class.dept
DEPT_NAME	Char(20)	Yes	Deps.dept_name
GROUP_NO	Integer(5)	Yes	Deps.Group_no
GROUP_NAME	Char(20)	Yes	Groups.group_name
DIVISION	Integer(5)	Yes	Groups.division
DIV_NAME	Char(20)	Yes	Division.div_name
COMPANY	Integer(5)	Yes	Comphead.company
CO_NAME	Char(20)	Yes	Comphead.co_name
PURCHASE_TYPE	Integer(1)	Yes	Deps.purchase_type

rmse_aip_orghier (RMS Extract of Organization Hierarchy to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_orghier.ksh

Design Overview

This script extracts from RMS organizational hierarchy information for integration with a time-phased inventory planning tool.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After dlyprg.pc and pre_rmse_aip.ksh.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
COMPHEAD	Yes	No	No	No
CHAIN	Yes	No	No	No
AREA	Yes	No	No	No
REGION	Yes	No	No	No
DISTRICT	Yes	No	No	No

I/O Specification

Output File Layout

The output file rmse_aip_orghier.dat is in fixed-length format matching to the schema definition in rmse_aip_orghier.schema.

Field Name	Field Type	Required	Description
DISTRICT	Integer(11)	No	District.district
DISTRICT_NAME	Char(20)	No	District.district_name
REGION	Integer(11)	No	Region.region
REGION_NAME	Char(20)	No	Region.region_name
AREA	Integer(11)	No	Area.area
AREA_NAME	Char(20)	No	Area.area_name
CHAIN	Integer(11)	Yes	Chain.chain
CHAIN_NAME	Char(20)	Yes	Chain.chain_name
COMPANY	Integer(5)	Yes	Comphead.company
CO_NAME	Char(20)	Yes	Comphead.co_name

rmse_aip_rec_qty (RMS Extract of Received PO and Transfer Quantities to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_rec_qty.ksh

Design Overview

This script extracts from RMS received PO and transfer quantities for integration with a time-phased inventory planning tool. Only records that meet the following criteria below are extracted:

For Purchase Orders:

- Ordhead.close_date is NULL or ordhead.close_date >= (current date - [1]max_notafter_days)
- Ordhead.not_after_date is not NULL
- Ordhead.orig_ind = 6 (external system generated)
- Ordloc.received_qty is not NULL

For Transfers:

- Tsfhead.close_date is NULL or tsfhead.close_date >= (current date - [1]max_notafter_days)
- Tsfhead.tsf_type = 'AIP' (generated by the time-phased inventory planning tool)
- Tsfhead.delivery_date is not NULL
- Tsfdetail.received_qty is not NULL

[1]*Defined in <etc_directory>/max_notafter_days.txt*

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After vrplbld.pc, cntrordb.pc, reqext.pc and pre_rmse_aip.ksh.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ORDHEAD	Yes	No	No	No
ORDLOC	Yes	No	No	No
ORDSKU	Yes	No	No	No
TSFHEAD	Yes	No	No	No
TSFDETAIL	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No

I/O Specification

Output File Layout

The output file received_qty.txt is in fixed-length format matching the schema definition in rmse_aip_rec_qty.schema.

Field Name	Field Type	Required	Description
ORDER_NUMBER	Integer(10)	Yes	Ordhead.order_no or tsfhead.tsf_no
ORDER_TYPE	Char(1)	Yes	'P' for purchase orders or 'T' for transfers
RMS_SKU	Char(25)	Yes	Ordsku.item or tsfdetail.item
ORDER_MULTIPLE	Integer(8)	Yes	Ordsku.supp_pack_size or tsfdetail.supp_pack_size
PACK_QTY	Integer(8)	Yes	If pack item then sum of V_packsku_qty.qty else 0
STORE	Integer(10)	No	If ordloc.loc_type = 'S' then ordloc.location or If tsfhead.to_loc_type = 'S' then tsfhead.to_loc
WAREHOUSE	Integer(10)	No	If ordloc.loc_type = 'W' then ordloc.location or If tsfhead.to_loc_type = 'W' then tsfhead.to_loc
RECEIVED_DATE	Date	Yes	Ordhead.not_after_date or tsfhead.delivery_date
QUANTITY	Integer(8)	Yes	Ordloc.qty_received or tsfdetail.received_qty

rmse_aip_store (RMS Extract of Stores to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_store.ksh

Design Overview

This script extracts RMS store information for integration with a time-phased inventory planning tool.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After storeadd.pc, likestore.pc, dlyprg.pc and pre_rmse_aip.ksh.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
STORE	Yes	No	No	No
STORE_FORMAT	Yes	No	No	No
CODE_DETAIL	Yes	No	No	No

I/O Specification

Output File Layout

The item output file is in fixed-length format matching to the schema definition in rmse_aip_store.schema.

Field Name	Field Type	Required	Description
STORE	Integer(11)	Yes	Store.store
STORE_NAME	Char(20)	Yes	Store.store_name
DISTRICT	Integer(11)	Yes	Store.district
STORE_CLOSE_DATE	Date	No	Store.store_close_date
STORE_OPEN_DATE	Date	Yes	Store.store_open_date
STORE_CLASS	Char(1)	Yes	Store.store_class
STORE_CLASS_DESCRIPTION	Char(40)	Yes	Code_detail.code_desc
STORE_FORMAT	Integer(5)	No	Store.store_format
FORMAT_NAME	Char(20)	No	Store_format.format_name
STOCKHOLDING_IND	Char(1)	Yes	Store.stockholding_ind
REMERCH_IND	Char(1)	Yes	Store.remerch_ind
CLOSING_STORE_IND	Char(1)	Yes	'N' if Store.store_close_date is empty, else 'Y'

rmse_aip_store_cur_inventory (RMS Extract of Store Current Inventory data to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_store_cur_inventory.ksh

Design Overview

This script extracts RMS current inventory for store locations for integration with a time-phased inventory planning tool. This script requires an 'F' or 'D' parameter:

- F - full extract of items/locations. Multiple output files. One file per item_loc_soh partition.
- D - delta extract of items/locations for the current day's transactions. Single output file.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	This program is run towards the end of the batch cycle where all inventory transactions are completed for the day. After stkvar.pc, wasteadj.pc, salstage.pc, reqext.pc and pre_rmse_aip.ksh.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	One thread per partition of item_loc_soh will be invoked if the script is run with a parameter of 'F'.

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ITEM_MASTER	Yes	No	No	No
ITEM_LOC_SOH	Yes	No	No	No
STORE	Yes	No	No	No
IF_TRAN_DATA	Yes	No	No	No
IF_TRAN_DATA_TEMP	Yes	Yes	No	No
PACKITEM	Yes	No	No	No
DBA_TAB_PARTITIONS	Yes	No	No	No

I/O Specification

Output File Layout

The output file sr0_curinv_{THREAD_NO}.txt is in fixed-length format matching the schema definition in rmse_aip_store_cur_inventory.schema.

Field Name	Field Type	Required	Description
STORE	Integer(20)	Yes	Item_loc_soh.loc
RMS_SKU	Char(20)	Yes	Item_master.item
STORE_CUR_INV	Integer(8)	Yes	Item_loc_soh.stock_on_hand – (item_loc_soh.tsf_reserved_qty + item_loc_soh.rtv_qty + item_loc_soh.non_sellable_qty + item_loc_soh.customer_resv + item_loc_soh.customer_backorder)

rmse_aip_substitute_items (RMS Extract of Substitute Items to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_substitute_item.ksh

Design Overview

This script extracts from RMS substitute item information for integration with a time-phased inventory planning tool.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After pre_rmse_aip.ksh.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
SUB_ITEMS_DETAIL	Yes	No	No	No

I/O Specification

Output File Layout

The output file rmse_aip_substitute_items.dat is in fixed-length format matching the schema definition in rmse_aip_substitute_items.schema.

Field Name	Field Type	Required	Description
ITEM	Char(25)	Yes	Sub_items_detail.item
LOCATION	Integer(10)	Yes	Sub_items_detail.location
SUB_ITEM	Char(25)	Yes	Sub_items_detail.sub_item
LOC_TYPE	Char(1)	Yes	Sub_items_detail.loc_type
START_DATE	Date	No	Sub_items_detail.start_date
END_DATE	Date	No	Sub_items_detail.end_date

rmse_aip_suppliers (RMS Extract of Supplier to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_suppliers.ksh

Design Overview

This script extracts from RMS supplier information for integration with a time-phased inventory planning tool. The script produces three extract files: rmse_aip_suppliers.dat, splr.txt and dmx_dirspl.txt. Splr.txt and dmx_dirspl.txt only contain active suppliers (sups.sup_status = 'A').

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After pre_rmse_aip.ksh.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
SUPS	Yes	No	No	No

I/O Specification

Output File Layout

The output file rmse_aip_suppliers.dat is in fixed-length format matching the schema definition in rmse_aip_suppliers.schema.

Field Name	Field Type	Required	Description
SUPPLIER	Integer(11)	Yes	Sups.supplier
SUP_NAME	Char(32)	Yes	Sups.sup_name

The output file splr.txt is in fixed-length format matching the schema definition in rmse_aip_splr.schema.

Field Name	Field Type	Required	Description
SUPPLIER	Integer(20)	Yes	Sups.supplier
SUPPLIER_DESCRIPTION	Char(40)	Yes	Sups.sup_name

The output file dmx_dirspl.txt is in fixed-length format matching the schema definition in rmse_aip_dmx_dirspl.schema.

Field Name	Field Type	Required	Description
SUPPLIER	Integer(20)	Yes	Sups.supplier
DIRECT_SUPPLIER	Char(1)	Yes	If sup.dsd_ind = 'Y' then 1, else if sup.dsd_ind = 'N' then 0

rmse_aip_tsf_in_well (RMS Extract of Transfers in the Well Quantities to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_tsf _in_well.ksh

Design Overview

This script extracts RMS "in the well" transfer quantities for integration with a time-phased inventory planning tool. In the well pertains to inventory that has been reserved by an approved or shipped transfer. The expected delivery date is also included in the extract.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After reqext.pc and pre_rmse_aip.ksh
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ITEM_MASTER	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
ITEM_SUPPLIER	Yes	No	No	No
TSFHEAD	Yes	No	No	No
TSFDETAIL	Yes	No	No	No
SHIPITEM_INV_FLOW	Yes	No	No	No
TRANSIT_TIMES	Yes	No	No	No
V_WH	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
PACKITEM	Yes	No	No	No

I/O Specification

Output File Layout

The output file rmse_aip_tsf_in_well.dat is in fixed-length format matching the schema definition in rmse_aip_tsf_in_well.schema.

Field Name	Field Type	Required	Description
DAY	Char(9)	Yes	Current date if tsfhead.delivery_date – transit_times.transit_time is less than current date else tsfhead.delivery_date – transit_times.transit_time
LOC	Integer(20)	Yes	If tsfhead.from_loc type = 'W' and tsfhead.tsf_type = 'EG' then shipitem_inv_flow.from_loc else tsfhead.from_loc
ITEM	Char(20)	Yes	Formal Case Type: If simple pack then and tsfhead.to_loc_type = 'S' then this would be the component of the pack in v_packsku_qty else item_master.item. Informal Case Type: Item_master.item
ORDER_MULTIPLE	Integer(4)	Yes	Formal Case Type: V_packsku_qty.qty for simple pack, else 1 Informal Case Type: One unique record for each item/supplier with order multiples of: 1, supp_pack_size, inner_pack_size and (ti * hi * supp_packsize)

Field Name	Field Type	Required	Description
TSF_RESERVED_QTY	Integer(8)	Yes	Formal Case Type: Tsfdetail.tsf_qty – tsfdetail.ship_qty. Resulting quantity is multiplied by V_packsku_qty.qty if item is a pack. Informal Case Type: Tsfdetail.tsf_qty – tsfdetail.ship_qty expressed in the primary case size. Remainder is in Standard UOM

rmse_aip_wh (RMS Extract of Warehouse to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_wh.ksh

Design Overview

This script extracts from RMS warehouse information for integration with a time-phased inventory planning tool. The script produces three extract files: rmse_aip_wh.dat, rmse_aip_wh.txt and rmse_aip_wh_type.txt. Only stock holding warehouses are extracted to the rmse_aip_wh.txt and rmse_aip_wh_type.txt files

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After whadd.pc and dlyprg.pc. After pre_rmse_aip.ksh.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
WH	Yes	No	No	No

I/O Specification

Output File Layout

The output file rmse_aip_wh.dat is in fixed-length format matching the schema definition in rmse_aip_wh_dat.schema.

Field Name	Field Type	Required	Description
WH	Integer(11)	Yes	Wh.wh
WH_NAME	Char(20)	Yes	Wh.wh_name
FORECAST_WH_IND	Char(1)	Yes	Wh.forecast_wh_ind
STOCKHOLDING_IND	Char(1)	Yes	Wh.stockholding_ind
WH_TYPE	Char(6)	No	Wh.vwh_type

The output file rmse_aip_wh.txt is in fixed-length format matching the schema definition in rmse_aip_wh.schema.

Field Name	Field Type	Required	Description
WAREHOUSE_CHAMBER	Char(20)	Yes	Wh.wh
WAREHOUSE_CHAMBER_ DESCRIPTION	Char(40)	Yes	Wh.wh_name
WAREHOUSE	Integer(20)	Yes	Wh.wh
WAREHOUSE_DESCRIPTION	Char(40)	Yes	Wh.wh_name

The output file rmse_aip_wh_type.txt is in fixed-length format matching the schema definition in rmse_aip_wh_type.schema.

Field Name	Field Type	Required	Description
WAREHOUSE	Integer(20)	Yes	Wh.wh
WH_TYPE	Char(6)	No	Wh.wh_type

rmse_aip_wh_cur_inventory (RMS Extract of Warehouse Current Inventory data to a Time-Phased Inventory Planning Tool)

Functional Area

RMS to time-phased inventory planning tool Integration

Module Affected

rmse_aip_wh_cur_inventory.ksh

Design Overview

This script extracts RMS current warehouse inventory information for integration with a time-phased inventory planning tool.

This script requires an 'F' or 'D' parameter:

- F - full extract of items/locations. Creates multiple files per warehouse. Files are concatenated into a single file upon successful completion.
- D - delta extract of items/locations for the current day's transactions. Creates a single extract file.

The script creates a backup of the previous day's data file labeled with the date on which they were created.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc Interface
Scheduling Considerations	After stkvar.pc, wasteadj.pc, salstage.pc, reqext.pc and pre_rmse_aip.ksh. After rmse_aip_store_cur_inventory.ksh if running a delta extract ('D' parameter).
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	One thread per warehouse will be invoked if the script is run with a parameter of 'F'.

Restart/Recovery

This is a standard Oracle Retail RETL script. No restart/recovery is used.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ITEM_MASTER	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No
ITEM_LOC_SOH	Yes	No	No	No
WH	Yes	No	No	No
ALLOC_DETAIL	Yes	No	No	No
ALLOC_HEADER	Yes	No	No	No
ORDHEAD	Yes	No	No	No
ITEM_SUPPLIER	Yes	No	No	No
PACKITEM	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
IF_TRAN_DATA_TEMP	Yes	No	No	No

I/O Specification

Output File Layout

The output file wr1_curinv.txt is in fixed-length format matching the schema definition in rmse_aip_wh_cur_inventory.schema.

Field Name	Field Type	Required	Description
WAREHOUSE	Integer(20)	Yes	Item_loc_soh.loc
RMS_SKU	Char(20)	Yes	Item_master.item
ORDER_MULT	Integer(4)	Yes	Formal Case Type: V_packsku_qty.qty for simple pack, else 1 Informal Case Type: One unique record for each item/supplier with order multiples of: 1, supp_pack_size, inner_pack_size and (ti * hi * supp_packsize)
WH_CUR_INV	Integer(8)	Yes	Formal Case Type: ((Item_loc_soh.stock_on_hand – (item_loc_soh.tsf_reserved_qty + item_loc_soh.rtv_qty + item_loc_soh.non_sellable_qty + item_loc_soh.customer_resv + item_loc_soh.customer_backorder)) - alloc_detail.qty_distro * (v_packsku_qty.qty for simple pack, else 1) Informal Case Type: ((Item_loc_soh.stock_on_hand – (item_loc_soh.tsf_reserved_qty + item_loc_soh.rtv_qty + item_loc_soh.non_sellable_qty + item_loc_soh.customer_resv + item_loc_soh.customer_backorder)) - alloc_detail.qty_distro)

5

Subscription Designs

PO Subscription API

Functional Area

PO subscription

Design Overview

RMS will expose an API that will allow external systems to create, edit, and delete purchase orders within RMS. The transaction will be performed immediately upon message receipt so success or failure can be communicated to the calling application.

Purchase order messages will be 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.

Consume Module

Filename: rmssub_xorders/b.pls

RMSSUB_XORDER.CONSUME

(O_status_code IN OUTVARCHAR2,
O_error_message IN OUTRTK_ERRORS.RTK_TEXT%TYPE,
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 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" 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 needs to 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.

For messages that creates or modifies purchase order, the ORDER_SETUP_SQL.POP_CONTAINER_ITEM() function is called. This function retrieves the container item for any deposit contents item in the message and adds them to the purchase order items. The order quantity of the container item is always equal to the order quantity of its contents item.

Once the message has been successfully persisted and processed for deposit container items, 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.

Business Validation Module

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,
I_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 results 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 a detail 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 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 and item/supp/country 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.
- 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.
- 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 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 and item/supp/country 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 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.
- 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 the order line is cancelled or reinstated via the indicators, calculate the new quantity buckets.
- 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.
- 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.
- Delete any allocations tied to the order line.
- Populate record ORDER_REC.ORDLOCS with the order/item/location for delete.

Bulk or Single DML Module

Filename: rmssub_xorders/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

ORDER MODIFY

- Updates a record in the ORDHEAD table.

ORDER DELETE

- Delete an order from ORDHEAD, ORDSKU, ORDLOC tables.

ORDER DETAIL CREATE

- Inserts records in the ORDLOC and optionally, ORDSKU tables

ORDER DETAIL MODIFY

- Updates records in the ORDLOC and/or ORDSKU 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.
- Delete the container items for any deposit contents items from ORDLOC and ORDSKU tables.
- Also verify it doesn't end up with an Approved order with no detail or with 0 total order quantity.

Message DTD

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

Message Types	Message Type Description	Document Type Definition (DTD)
XorderCre	Order Create Message	XOrderDesc.dtd
XorderMod	Order Modify Message	XOrderDesc.dtd
XorderDel	Order Delete Message	XOrderRef.dtd
XorderDtlCre	Order Detail Create Message	XOrderDesc.dtd
XorderDtlMod	Order Detail Modify Message	XOrderDesc.dtd
XorderDtlDel	Order Detail Delete Message	XOrderRef.dtd

Design Assumptions

Required fields are shown in mapping document.

Many ordering functionalities that are available on-line are not supported via this API. See the SAE and the story document for a list of these. Triggers related to these functionalities should be turned off. Oracle Retail 11 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.

Tables

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_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

TABLE	SELECT	INSERT	UPDATE	DELETE
ALLOC_DETAIL	Yes	No	No	Yes
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

Transfer Subscription

Functional Area

Transfer subscription

Design Overview

RMS subscribes to transfers from external subsystems that provide only basic information about these transactions, namely item, supplier, location and quantity.

It will be expected that RMS users will be responsible for updating and monitoring the financial and execution data such as transfer costs.

RMS monitors all of the shipments and receipts and will close the transfer once the received quantity equals the quantity requested or an outside system cancels the outstanding quantity. RMS will maintain the perpetual inventory for each location as it currently does.

The transfer RIB API will have 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.

Consume Module

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 will need to initially ensure that the passed in message type is a valid type for transfer 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. 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 needs to verify that the message passes all of the RMS 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.

Business Validation Module

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 results 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 create or a detail modify message, only the TSF number will be validated at the header level; all other header fields are ignored.

TRANSFER CREATE

- Check required fields.
- Validate fields.
- Verify that the delivery date is not NULL if RMS is integrated with AIP (Advanced Inventory Planning).
- Default fields (status at header,freight type and tsf type)
- Build transfer records.

TRANSFER MODIFY

- Check required fields on the header nodes.
- Verify TSF number already exists.
- Verify that the delivery date is not NULL if RMS is integrated with AIP (Advanced Inventory Planning).
- Validate fields.
- Populate record.

TRANSFER DETAIL CREATE

- Check required fields on the detail node.
- Verify TSF number already exists.
- Verify tsf/item/loc does not already exist.
- 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.
- Populate record.

TRANSFER DETAIL MODIFY

- Check required fields on the detail node.
- Verify transfer/item/loc already exists.
- If TSF quantity is reduced, verify the new quantity is not below what has already been received plus what is being shipped or expected.
- Populate 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

- Check required fields.
- Verify TSF number already exists.
- Verify that TSF is not already shipped or received.
- Populate record for delete.

TRANSFER DETAIL DELETE

- Check required fields.
- Verify TSF/item/loc already exists.
- Verify that TSF line is not already shipped or received.
- Populate record with the TSF no/item/location for delete.

Bulk or Single DML Module

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

- Delete 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 DTD

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	Document Type Definition (DTD)
xtsfcre	Transfer Create Message	XTsfDesc.dtd
xtsfmod	Transfer Modify Message	XTsfDesc.dtd
xtsfdel	Transfer Delete Message	XTsfRef.dtd
xtsfdtlcre	Transfer Detail Create Message	XTsfDesc.dtd
xtsfdtlmod	Transfer Detail Modify Message	XTsfDesc.dtd
xtsfdtlcel	Transfer Detail Delete Message	XTsfRef.dtd

Design Assumptions

Required fields are shown in RIB documentation.

Tables

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

6

Batch Designs

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

Distro Price Change Publish [distropcpub]

Functional Area

Pricing/Transfers/Allocations

Module Affected

DISTROPCPUB.PC

Design Overview

The DISTROPCPUB.PC program will get price change information for any allocations and transfers and write the information to the corresponding queue table. This program will ensure that Oracle Retail Warehouse Management will have access to any item/location unit retail information that is changed after an allocation or transfer has been published.

This program will loop through the PRICE_HIST table, selecting records whose unit retail will change for vdate+1, and transaction type is in 4 (single unit retail was changed) or 11(single unit retail and multi-unit retail were changed). It will then search for allocations and transfers with matching item/locations. When a match is found, depending on the distro type, the program calls allocation or transfer publishing logic to insert the data into the allocation or transfer queue table, so that the RIB can publish the change to the warehouse system.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	PHASE 3 – Daily
Scheduling Considerations	This program should run after RPM price event execution batch process.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	Multithreading based on store.

Restart/Recovery

The logical unit of work is store. The driving cursor retrieves all item/locations that have price changes in effect from the next day. It also gets all of the component items of the non-sellable packs that have price changes.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
PERIOD	Yes	No	No	No
PRICE_HIST	Yes	No	No	No
V_RESTART_STORE	Yes	No	No	No
V_PACKSKU_QTY	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
ALLOC_HEADER	Yes	No	No	No
ALLOC_DETAIL	Yes	No	No	No
TSFHEAD	Yes	No	No	No
TSFDETAIL	Yes	No	No	No
ORDHEAD_REV	Yes	No	No	No
ORDHEAD	Yes	No	No	No
ALLOC_MFQUEUE	No	Yes	No	No
TSF_MFQUEUE	No	Yes	No	No

Shared Modules

N/A

I/O Specification

N/A

EDI Location Address to Vendor Download [edidladd]

Design Overview

The purpose of this module is to download addresses of stores and warehouses to vendors. The address fields in the tables store and wh were removed and were referenced instead to the table addr.

The output file format will be a standard Oracle Retail file format that will be translated into EDI format by the Gentran translator. Addresses will be downloaded in two different scenarios. The program will download changes made to store or warehouse addresses into a flat file. Further, if a system_options table flag (addr_catalog) is set to true (Y), then the addresses of all stores and warehouses will be downloaded into a different file to be sent to suppliers. This program will run nightly.

When a store or warehouse address is changed a flag will be set to true to indicate that that location should have its address changes submitted to suppliers. When the changes occur on the location forms (store.fmb or wh.fmb form), the addr_change field is set to true (Y) on the location tables (store or wh). After each changed location address is processed by the EDIDL838.pc program its addr_change flag is reset to N.

The addr_catalog flag for sending address catalogs is set on the sys_ctrl.fmb form. When this is set a catalog listing of all store and warehouse addresses are sent to suppliers, then the addr_catalog field on system_options will be reset to N.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ADDR	No	Yes	No	Yes	No
ADD_TYPE	No	Yes	No	No	No
ADD_TYPE_MODULE	No	Yes	No	No	No
PERIOD	No	Yes	No	No	No
STORE	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	Yes	No
WH	No	Yes	No	No	No

Stored Procedures / Shared Modules (Maintainability)

None

Program Flow

N/A

Function Level Description

Init()

Get system variables: current date and catalog indicator flag. Open output files: for location address changes, address change rejects, canonical catalog listing, and catalog listing rejects. (only open catalog and catalog reject files if a catalog is to be written). Call restart_init. Set up format strings for ouput records.

Process()

For each record fetched from the driving cursor, call a function to process location address changes, and if the catalog indicator flag is set to true to write catalog listing of all locations.

Write FTAIL record to file.

The column addr_catalog in table system_options is then updated to 'N' if its previous value was 'Y'.

Write_records_loc()

This function should write out the address change record to the output file and update the flag on the location table.

Write_records_cat()

This function writes address change records to the catalog file.

Final()

Call restart/recovery close.

I/O Specification

Output file format (all characters should be right-padded with blanks and left justified; all numbers should be left-padded with zeroes and right justified)

Record Name	Field Name	Field Type	Default Value	Description
File header	File type record descriptor	Char(5)	FHEAD	Identifies file record type
	File line sequence	Number(10)	Start at 1 and increment	Line number of file
	File type definition	Char(5)	DLADD	Identifies file source
	Purpose code	Char(2)	04 (change) or 05 (replace)	Add/change location or replace whole list
Transaction Detail	File type record descriptor	Char(5)	TDETL	Identifies file record type
	File line sequence	Number(10)	Increment	Line number of file
	Transaction number	Number(10)	Start at 1,increment	Identifies transaction
	Date	Char(8)		Period.vdate YYYYMMDD
	Store or warehouse	Char(2)	SN (store) or WH (warehouse)	Location type
	Location	Number(10)		Store.store or wh.wh
	Location name	Char(20)		Store..store_name or wh.wh_name
	Address line 1	Char(30)		Store.store_add1/wh.wh_add1
	Address line 2	Char(30)		Store_add2 or wh_add2
	City	Char(20)		Store.store_city/wh.wh_city
	State	Char(3)		Store.state/wh.state
	Postal code	Char(10)		Store_pcode/wh_pcode
	Country	Char(3)		Store/wh.country_id
	Address Type Description	Char(40)		atp.type_desc

Record Name	Field Name	Field Type	Default Value	Description
File trailer	File type record descriptor	Char(5)	FTAIL	Identifies file record type
	Total number lines	Number(10)		Total lines in file
	Total no. TDETL lines	Number(10)		(total lines – 2)

Restart/Recovery

Because of the lack of volume and the flexibility requirements of EDI, the program will use Oracle Retail's standard restart/recovery only minimally. The driving query volume is limited to the volume of the store and warehouse tables. Further, the output files that are created are created if they don't exist and are overwritten if they already exist. In the event of a fatal error it is, therefore, reasonable to expect clients to simply restart the job from the beginning without recovery.

Driving cursor:

```
    SELECT :ps_str_type,
           s.store,
           s.store_name,
           a.add_1,
           a.add_2,
           a.city,
           a.state,
           a.post,
           a.country_id,
           a.edi_addr_chg,
           atp.type_desc,
           ROWIDTOCHAR(a.rowid)
      FROM store           s,
           addr            a,
           add_type_module atm,
           add_type        atp
     WHERE a.addr_type        = atm.address_type
       AND a.module           = atm.module
       AND atm.address_type   = atp.address_type
       AND a.key_value_1      = to_char(s.store)
       AND a.edi_addr_chg     = DECODE(:pi_do_all,1,a.edi_addr_chg,'Y')
       AND a.primary_addr_ind = 'Y'
       AND a.module           = 'ST'
UNION ALL
   SELECT :ps_wh_type,
          w.wh,
          w.wh_name,
          a.add_1,
          a.add_2,
          a.city,
          a.state,
          a.post,
          a.country_id,
          a.edi_addr_chg,
          atp.type_desc,
          ROWIDTOCHAR(a.rowid)
     FROM wh              w,
          addr            a,
          add_type_module atm,
```

```
      add_type        atp
WHERE a.addr_type        = atm.address_type
  AND a.module           = atm.module
  AND atm.address_type   = atp.address_type
  AND a.key_value_1      = to_char(w.wh)
  AND a.edi_addr_chg     = DECODE(:pi_do_all,1,a.edi_addr_chg,'Y')
  AND a.primary_addr_ind = 'Y'
  AND a.module           = 'WH';
```

This selects all location information if the catalog is to be processed but only information from locations where the addr_chg indicator is set to Y if the catalog shouldn't be processed. Pi_do_all will be an integer variable that should be initialized as zero and set to 1 just before calling process_catalog.

Scheduling Constraints

Processing Cycle: PHASE 4 (DAILY)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

EDI Supplier Address Upload [ediupadd]

Design Overview

The ediupadd.pc batch program will read vendor/supplier sent EDI 838 Profile Data Files. These files will be processed by vendor/supplier and used to update Oracle Retail supplier address information.

Five different types of supplier addresses can be changed via this EDI interface: business, postal, returned to, pick up and payment mailing address. This program always assumes that address information is primary for the address type.

If there is an error with a transaction set, write that transaction to the reject file so it can be fixed and reprocessed later, then the program will continue to the next transaction set. See EDI856 for an example.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
STATE	No	Yes	No	No	No
SUPPLIER	No	Yes	No	No	No
SUPS_ADD	No	No	Yes	Yes	No

Scheduling Constraints

Processing Cycle: Ad Hoc

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (file based processing—single thread only)

Restart Recovery

Oracle Retail restart/recovery capability is minimal. The program uses non-fatal error handling to process input files. There is not enough volume to warrant the use of restart recovery. A commit will not occur until the end of file processing and therefore if fatal errors are encountered updates will not have been committed and the program can be restarted without recovery.

Program Flow

N/A

Shared Modules

N/A

Function Level Description

Init()

Open input & output files. Validate file header line. Call Oracle Retail API restart_init to initializse restart/recovery process.

Process()

Loop through each line of input file. Call validate_FDETL to validate and format variables.. Write to reject file if variables are not valid. Call update_supplier to update the database with the new addresses.

Update_supplier()

Update sups_add table with new, modified address information. This program presumes that address modification information is regarding a "primary" address. If the action specified is "add", then any existing primary address of that type will be changed so that it is no longer primary, and the new address will be inserted as a primary address of the specified type into the supplier address table. If the action is "update", then the existing primary address of the specified type will be modified to the new passed information (if one does not exist for that type, it will be inserted).

validate_FDETL—validate supplier and address information

I/O Specification

Input file:
FHEAD File type identification
FDETL Supplier address info
FTAIL End of file marker

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	**File record descriptor**	**Char(5)**	**FHEAD**	**Describes file line type**
	Line number	Number(10)		Sequential file line number
	Gentran_id	Char(5)		Identifies the file type
	File create date	Char(14)		YYYYMMDDHH24MISS format
FDETL	**File record descriptor**	**Char(5)**	**FDETL**	**Describes file line type**
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Add or Update	Char(1)		'A'dd or 'U'pdate address
	Address type	Char(2)		Will be translated into sups_add.address_type: 01 – Business 02 – Postal 03 – returns 04 – Pick Up (Order) 05 – Payment
	Supplier	**varChar(10)**		Sups.supplier
	Address line 1	Char(30)		Sups_add.address_1
	Address line 2	Char(30)		Sups_add.address_2
	Address line 3	Char(30)		Sups_add.address_3
	Contact name	Char(20)		Sups_add.contact_name
	Contact Phone	Char(20)		Sups_add.contact_phone
	Contact fax	Char(20)		Sups_add.contact_fax
	City	Char(20)		Sups_add.city

Record Name	Field Name	Field Type	Default Value	Description
	State	Char(3)		Sups_add.state
	Postal code	Char(10)		Sups_add.post_code
	Country	Char(3)		Sups_add.country_id
FTAIL	**File record descriptor**	**Char(5)**		**Describes file record type**
	Line number	Number(10)		Sequential file line number (total # lines in file)
	Number of transactions	Number(10)		Number of transactions in file

Technical Issues

N/A

Oracle Retail Demand Forecasting Purge [fcstprg]

Functional Area

Demand Forecasting

Module Affected

FCSTPRG.PC

Design Overview

This program deletes data from forecast information tables. Data deletion is performed by partition truncation, table truncation or deletion by domain. The method of deletion is dependent on whether or not the table is partitioned. This program serves to delete data by domains so that they can re-loaded with new forecast information from RDF.

This program must be run as either the RMS schema owner, or be run by a user that has been granted the following system privileges:

- 'drop any table'
- 'alter any table'

Scheduling Constraints

Schedule Information	Description
Processing Cycle	AD-HOC
Scheduling Considerations	N/A
Pre-Processing	prepost fcstprg pre - disables indexes
Post-Processing	prepost fcstprg post - rebuilds indexes
Threading Scheme	N/A

Restart/Recovery
N/A

Locking Strategy
N/A

Security Considerations
N/A

Performance Considerations
N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
ITEM_FORECAST	No	No	No	Yes
DEPT_SALES_FORECAST	No	No	No	Yes
CLASS_SALES_FORECAST	No	No	No	Yes
SUBCLASS_SALES_FORECAST	No	No	No	Yes

I/O Specification
N/A

Oracle Retail Demand Forecasting Rollup [fcstrbld]

Functional Area
Demand Forecasting

Module Affected
FCSTRBLD.PC

Design Overview

This program is designed to roll-up new or updated forecasted unit sales data from the item_forecast table. This data will be summarized into the subclass, class and department level sales forecast tables.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	PHASE 3 (weekly)
Scheduling Considerations	N/A
Pre-Processing	N/A
Post-Processing	prepost fcstrbld post – truncates the FORECAST_REBUILD table
Threading Scheme	Threaded by domain id

Restart/Recovery

The logical unit of work is a domain id. The program commits each time the rollups (dept, class and subclass) for a domain id is successfully processed.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
FORECAST_REBUILD	Yes	No	No	Yes
SUBCLASS_SALES_FORECAST	Yes	Yes	No	No
ITEM_MASTER	Yes	No	No	No
ITEM_FORECAST	Yes	No	No	No
STORE	Yes	No	No	No
CLASS_SALES_FORECAST	Yes	Yes	No	No
DEPT_SALES_FORECAST	Yes	Yes	No	No

I/O Specification

N/A

Oracle Retail Demand Forecasting Rollup by Department, Class and Subclass [fcstrbld_sbc]

Functional Area

Demand Forecasting

Module Affected

FCSTRBLD_SBC.PC

Design Overview

The module rolls up the sales forecast data at subclass and class level to class and department level respectively and inserts the data. The program selects records from the table SUBCLASS_SALES_FORECAST and writes the records to CLASS_SALES_FORECAST and selects the data from CLASS_SALES_FORECAST and writes into DEPT_SALES_FORECAST using the domain ID stored in the table FORECAST_REBUILD. The record in FORECAST_REBUILD is deleted after the record is written to the above destination tables.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Phase 3 (Weekly)
Scheduling Considerations	After completion of FCSTRBLD.PC.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

Restart/recovery is based on the values stored in restart_bookmark from the last commit prior to failure. The values are for the last domain_id that was not rolled up completely.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
FORECAST_REBUILD	Yes	No	No	Yes
CLASS_SALES_FORECAST	Yes	Yes	No	No
DEPT_SALES_FORECAST	No	Yes	No	No
SUBCLASS_SALES_FORECAST	Yes	No	No	No
STORE	Yes	No	No	No

I/O Specification

N/A

Geocode Hierarchy Upload [gcupld]

Functional Area

Geocode hierarchy

Module Affected

GCUPLD.PC

Design Overview

A geocode identifies a combination of the country, state, county and city in which locations operate.

GCUPLD.PC (geocode hierarchy upload) provides the ability to upload geocodes from an outside source into RMS. This batch module lets retailers delete current geocodes and create new geocodes in the system. A flat file is used to feed the program the additions and deletions to the geocode tables. Validation determines if duplicate records exist, dependencies exist, and the flat file is in the correct format. If errors occur in the validation of the record, it is written out to a reject file to allow further investigation of the record.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad Hoc
Scheduling Considerations	Ad Hoc
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

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

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
GEOCODE_TEMP	YES	YES	NO	YES
DISTRICT_GEOCODES	YES	YES	NO	YES
CITY_GEOCODES	YES	YES	NO	YES
COUNTY_GEOCODES	YES	YES	NO	YES
STATE_GEOCODES	YES	YES	NO	YES
COUNTRY_GEOCODES	YES	YES	NO	YES
GEOCODE_STORE	YES	NO	NO	NO
GEOCODE_TXCDE	YES	NO	NO	NO

I/O Specification

Output File Layout

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	File head descriptor	Char(5)	FHEAD	Describes the file line type
	Line id	Char(10)	0000000001	Sequential file line number
	Gentran ID	Char(4)	'GCUP'	Identifies which translation Gentran uses
	Current date	Char(14)		File date in YYYYMMDDHH24MISS format
FDETL	File record descriptor	Char(5)	FDETL	Describes file line type
	Line id	Char(10)		Sequential file line number
	Country Geocode	Char(4)		Country Geocode

Record Name	Field Name	Field Type	Default Value	Description
	State Geocode	Char(4)		State Geocode
	County Geocode	Char(4)		County Geocode
	City Geocode	Char(4)		City Geocode
	District Geocode	Char(4)		District Geocode
	Geocode Level	Char(6)		Geocode Level Valid values are: 'CNTRY,'STATE','COUNTY', 'CITY', 'DIST'
	Geocode Description	Char(40)		Geocode Description
	Add Delete Ind	Char(1)		Add/delete Indicator Valid values are: 'A', 'D'
FTAIL	File record descriptor	Char(5)	FTAIL	Marks end of file
	Line id	Char(10)		Sequential file line number
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

Inventory Adjustment Purge [invaprg]

Functional Area

Inventory Adjustment

Module Affected

INVAPRG.PC

Design Overview

The Inventory Adjustment Purge module deletes all obsolete inventory adjustment records whose adjustment date has elapsed by a pre-determined number of months. The number of months that inventory adjustment records are kept before they are purged by this batch is defined in the SYSTEM_OPTIONS table.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	AD-HOC (monthly)
Scheduling Considerations	N/A
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery
N/A

Locking Strategy
N/A

Performance Considerations
N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
UNIT_OPTIONS	Yes	No	No	No
PERIOD	Yes	No	No	No
INV_ADJ	No	No	No	Yes

Shared Modules
N/A

I/O Specification
N/A

End Of Year Inventory Position Purge [nwppurge]

Functional Area
Stock Ledger

Module Affected
NWPPURGE.PC

Design Overview
This program purges the records from the table NWP after a certain amount of years have passed. The number of years is a configurable parameter setup in SYSTEM_OPTIONS.nwp_retention_period.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Ad-Hoc
Scheduling Considerations	N/A
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	N/A

Restart/Recovery

Restart/recovery is not applicable, but the records will be committed based on the commit max counter setup in the restart control table.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
NWP	Yes	No	No	Yes

I/O Specification

N/A

End of Year Inventory Position Snapshot [nwpyearend]

Functional Area

Stock count

Module Affected

NWPYEAREND.PC

Design Overview

This program takes a snapshot of the item's stock position and cost at the end of the year. When the end of year NWP snapshot process runs, it takes a snapshot of stock and weighted average cost (WAC) for every item/location combination currently holding stock. If there is not a record already on the NWP table for an item/location/year combination in the snapshot, a new record is added for that item/location/year combination.

Scheduling Constraints

Schedule Information	Description
Processing Cycle	Phase 4 (Yearly)
Scheduling Considerations	Needs to run on the last day of the year in phase 4.
Pre-Processing	N/A
Post-Processing	N/A
Threading Scheme	Multithreaded by store_wh

Restart/Recovery

The logical unit of work for this program is set at the location/item level. Threading is done by supplier using the v_restart_store_wh view to thread properly.

The commit_max_ctr field should be set to prevent excessive rollback space usage, and to reduce the overhead of file I/O. The changes will be posted when the commit_max_ctr value is reached and the value of the counter is subject to change based on implementation.

Locking Strategy

N/A

Security Considerations

N/A

Performance Considerations

N/A

Key Tables Affected

Table	Select	Insert	Update	Delete
NWP_FREEZE_DATE	Yes	No	No	No
ITEM_MASTER	Yes	No	No	No
NWP	Yes	Yes	Yes	No
ITEM_LOC_SOH	Yes	No	No	No

I/O Specification

N/A

Sales Audit Get Reference [sagetref]

Design Overview

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

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ITEM_MASTER	No	Yes	No	No	No
ITEM_LOC	No	Yes	No	No	No
VAR_UPC_EAN	No	Yes	No	No	No
SA_IMPORT_LOG	No	Yes	No	No	No
CURRENCIES	No	Yes	No	No	No
STORE_DAY	No	Yes	No	No	No
STORE	No	Yes	No	No	No
SA_STORE_DAY	No	Yes	No	No	No
CODE_DETAIL	No	Yes	No	No	No
SA_ERROR_CODES	No	Yes	No	No	No

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
SA_CC_VAL	No	Yes	No	No	No
SA_STORE_POS	No	Yes	No	No	No
POS_TENDER_TYPE_HEAD	No	Yes	No	No	No
NON_MERCH_CODE_HEAD	No	Yes	No	No	No
PARTNER	No	Yes	No	No	No
SUPS	No	Yes	No	No	No
SA_STORE_EMP	No	Yes	No	No	No
BANNER	No	Yes	No	No	No
CHANNELS	No	Yes	No	No	No
ADDR	No	Yes	No	No	No

Stored Procedures / Shared Modules (Maintainability)

N/A

Function Level Description

main()

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

init()

This function will initialize the necessary restart recovery variables.
Calls the function retek_init().

process()

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

process_item_master_info()

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

size_item_master_arrays()

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

write_item_data()

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

write_waste_data()

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

process_ref_item_info()

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

size_ref_item_arrays()

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

write_ref_item_data()

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

process_prim_variant_info()

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

size_prim_variant_arrays()

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

write_prim_variant_data()

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

process_var_upc_ean_info()

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

size_var_upc_ean_arrays()

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

write_var_upc_data()

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

process_store_day_info()

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

size_store_day_arrays()

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

write_store_day_data()

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

process_codes_info()

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

size_codes_arrays()

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

write_codes_data()

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

process_error_info()

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

size_error_arrays()

This function allocates memory for the error array used in process_error_info.

write_error_data()

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

process_cc_val_info()

This function will query credit card validation information from the sa_cc_val table and uses this information to populate the credit card validation array. This function also calls size_cc_val_arrays() to allocate memory for the credit card validation array. The columns selected in this process include length, from_prefix, to_prefix, tender_type_id, and value type. The information should be ordered by length (alphabetically not numerically) and from_prefix. All records in the credit card validation array should be written to the credit card validation output file by calling write_cc_val_data().

size_cc_val_arrays()

This function allocates memory for the credit card valdiation array used in process_cc_val_info.

write_cc_val_data()

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

process_store_pos_info()

This function will query store POS information from the sa_store_pos table and uses this information to populate the store POS array. This function also calls size_store_pos_arrays() to allocate memory for the store POS array. The columns selected in this process include store, pos_type, start_tran_no, and end_tran_no. The information should be ordered by store (alphabetically not numerically) and pos_type. All records in the store POS array should be written to the store POS output file by calling write_store_pos_data().

size_store_pos_arrays()

This function allocates memory for the store POS array used in process_store_pos_info.

write_store_pos_data()

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

process_tender_type_info()

This function will query tender type information from the pos_tender_type_head table and uses this information to populate the tender type array. This function also calls size_tender_type_arrays() to allocate memory for the tender type array. The columns selected in this process include tender_type_group, tender_type_id, and tender_type_desc. The information should be ordered by tender_type_group and tender_type_id (alphabetically not numerically). All records in the tender array should be written to the tender type output file by calling write_tender_type_data().

size_tender_type_arrays()

This function allocates memory for the tender type array used in process_tender type_info.

write_tender_type_data()

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

process_merch_codes_info()

This function will query merch code information from the non_merch_code_head table and uses this information to populate the merch codes array. This function also calls size_merch_codes_arrays() to allocate memory for the merch codes array. The columns selected in this process include non_merch_code. The information should be ordered by non_merch_code. All records in the merch codes array should be written to the merchant codes output file by calling write_merch_codes_data().

size_merch_codes_arrays()

This function allocates memory for the merch codes array used in process_merch_codes_info.

write_merch_codes_data()

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

process_partner_info()

This function will query partner information from the partner table and uses this information to populate the partner array. This function also calls size_partner_arrays() to allocate memory for the partner array. The columns selected in this process include partner_type, and partner_id. The information should be ordered by partner_id. All records in the partner array should be written to the partner output file by calling write_partner_data().

size_partner_arrays()

This function allocates memory for the partner array used in process_partner_info.

write_partner_data()

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

process_supplier_info()

This function will query supplier information from the sups table and uses this information to populate the supplier array. This function also calls size_supplier_arrays() to allocate memory for the supplier array. The columns selected in this process include supplier, and sups_status. The information should be ordered by supplier (alphabetically not numerically). All records in the supplier array should be written to the supplier output file by calling write_supplier_data().

size_supplier_arrays()

This function allocates memory for the supplier array used in process_supplier_info.

write_supplier_data()

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

process_employee_info()

This function will query employee information from the sa_store_emp table and uses this information to populate the employee array. This includes all stores where pos_id is NOT NULL. This function also calls size_employee_arrays() to allocate memory for the employee array. The columns selected in this process include store, pos_id, and emp_id. The information should be ordered by store (alphabetically not numerically) and pos_id. All records in the employee array should be written to the employee output file by calling write_employee_data().

size_employee_arrays()

This function allocates memory for the employee array used in process_employee_info.

write_employee_data()

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

process_banner_info()

This function will query banner information from the store, banner and channels tables and uses this information to populate the banner array. This function also calls size_banner_arrays() to allocate memory for the banner array. The columns selected in this process include store, and banner_id. The information should be ordered by store (alphabetically not numerically). All records in the banner array will be written to the banner output file by calling write_banner_data().

size_banner_arrays()

This function allocates memory for the banner array used in process_banner_info.

write_banner_data()

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

final()

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

Input Specifications

N/A

Output Specifications

As all files produced by this program are used only internally, they consist of only detail records.

Char field types are left justified and blank padded.

Number field types are right justified and zero padded.

Record Name	Field Name	Field Type	Default Value	Description
Item Data	Item	char(25)		Unique item identifier
	Dept	char(4)		Department identifier
	Class	char(4)		Class identifier
	Subclass	char(4)		Subclass identifier
	Standard_uom	char(4)		Standard UOM
	Catch_weight_ind	char(1)		Catch weight indicator
Waste Data	Item	char(25)		Unique item identifier
	Waste_type	char(6)		Waste type identifier
	Waste_pct	number(16)		Waste percent
Ref Item Data	Item_parent	char(25)		Item Parent
	Item	char(25)		Unique item identifier

Record Name	Field Name	Field Type	Default Value	Description
Primary Variant Data	Item_loc	number(10)		Item location
	Item	char(25)		Unique item identifier
	Primary_variant	char(25)		Primary variant
Variable UPC Data	Format_id	char(1)		Format identifier
	Format_desc	char(20)		Format description
	Prefix_length	number(1)		Prefix length
	Begin_item_digit	number(2)		Determines the first digit of the item number.
	Begin_var_digit	number(2)		Determines the first digit of the variable weight/price.
	Check_digit	number(2)		Position of the check digit.
	Prefix	number(2)		Item master prefix
Store Day Data	Store	number(10)		Store number
	Business_date	char(8)		Business date – format: YYYYMMDD
	Store_day_seq_no	number(20)		Unique store/day identifier
	Day	number(3)		Day
	Tran_no_generated	char(6)		If NULL then blank
	System_code	char(6)		System code
	Currency_rtl_dec	number(1)		Currency retail decimal places
Code Data	Code_type	char(4)		Unique code type identifier
	Code	char(6)		Unique code identifier
	Code_seq	number(4)		Unique code sequence identifier
Error Data	Error_code	char(25)		Error identifier
	Error_desc	char(255)		Error description
	Rec_solution	char(255)		Recommended solution (If NULL then 'there is no solution')
Credit Card Validation Data	Length	number(2)		Card number length
	From_prefix	number(6)		Start value for range of valid prefixes.

Record Name	Field Name	Field Type	Default Value	Description
	To_prefix	number(6)		End value for range of valid prefixes.
	Tender_type_id	number(6)		Credit card ID
	Val_type	char(6)		Validation type. If NULL, than use "NONE".
Store POS Data	Store	number(10)		Store identifier
	Pos_type	char(6)		POS type identifier
	Start_tran_no	number(10)		First transaction number produced. Right justified and zero padded.
	End_tran_no	number(10)		Last transaction number produced. Right justified and zero padded.
Tender Type Data	Tender_type_group	char(6)		Tender type group
	Tender_type_id	number(6)		Tender type identifier. Right justified and zero padded.
	Tender_type_desc	char(40)		Tender type description.
Merchant Code Data	Non_merch_code	char(6)		Code identifying a non-merchandise cost that can be added to an invoice.
Partner Data	Partner_type	char(6)		Specifies the type of partner. Valid values are Bank 'BK', Agent 'AG', Freight Forwarder 'FF', Importer 'IM', Broker 'BR', Factory 'FA', Applicant 'AP', Consolidator 'CO', and Consignee 'CN', Supplier hierarchy level 1 'S1', Supplier hierarchy level 2 'S2', Supplier hierarchy level 3 'S3'.
	Partner_id	char(10)		Partner vendor number.
Supplier Data	Supplier	number(10)		Supplier vendor number.
	Sup_status	char(1)		Determines whether the supplier is currently active. Valid values include: 'A' for an active supplier or 'I' for an inactive supplier.
Employee Data	Store	number(10)		Store number.
	Pos_id	char(10)		The POS ID of the employee.
	Emp_id	char(10)		The employee ID of the employee.

Record Name	Field Name	Field Type	Default Value	Description
Banner Data	Store	number(10)		Store number
	Banner_id	number(4)		Banner identifier

Scheduling Considerations

- Processing Cycle: Anytime – Sales Audit is a 24/7 system.
- Scheduling Diagram: This module should be executed in the earliest phase, before the first import of RTLOGs into ReSA.
- Pre-Processing: sastdycr.pc
- Post-Processing: saimptlog.pc
- Threading Scheme: N/A

Restart Recovery

Restart recovery does not apply in the typical sense because sagetref writes to output files and will not need to have restart capabilities, however restart is used for bookmarking purposes.