Oracle® Retail Merchandising System

Operations Guide, Volume 3 - Back End Configuration and Operations
Release 13.2.3

October 2011

ORACLE

Oracle® Retail Merchandising System Operations Guide, Volume 3 - Back End Configuration and Operations, Release 13.2.3

Primary Author: Nathan Young

Value-Added Reseller (VAR) Language

Oracle Retail VAR Applications

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

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

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

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

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

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

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

Contents

Send Us Your Comments

Oracle Retail Merchandising System Operations Guide, Volume 3 - Back End Configuration and Operations, Release 13.2.3

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

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

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

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

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

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

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

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

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

Preface

This operations guide serves as an Oracle Retail Merchandising System (RMS) solution reference to explain 'backend' processes and configuration.

Audience

Anyone who has an interest in better understanding the inner workings of the Retail Merchandising System can find valuable information in this guide. There are three audiences in general for whom this guide is written:

- System analysts and system operation personnel:
  - Who are looking for information about Retail Merchandising System processes internally or in relation to the systems across the enterprise
  - Who operate the Retail Merchandising System on a regular basis
- Integrators and implementation staff who have the overall responsibility for implementing the Retail Merchandising System in their enterprise
- Business analysts who are looking for information about processes and interfaces to validate the support for business scenarios within the Retail Merchandising System and other systems across the enterprise

Related Documents

For more information, see the following documents:

- *Oracle Retail Merchandising System Release Notes*
- *Oracle Retail Merchandising System Installation Guide*
- *Oracle Retail Merchandising System Operations Guide*
- *Oracle Retail Merchandising System Custom Flex Attribute Solution Implementation Guide*
- *Oracle Retail Merchandising System Data Model*
- *Oracle Retail POS Suite 13.3.3/Merchandising Operations Management 13.2.3 Implementation Guide*
- *Oracle Retail Merchandising Batch Schedule*
- *Oracle Retail Merchandising Implementation Guide*

Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:

https://support.oracle.com

When contacting Customer Support, please provide the following:

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

Review Patch Documentation

When you install the application for the first time, you install either a base release (for example, 13.2) or a later patch release (for example, 13.2.3). If you are installing the base release and additional patch and bundled hot fix releases, read the documentation for all releases that have occurred since the base release before you begin installation. Documentation for patch and bundled hot fix releases can contain critical information related to the base release, as well as information about code changes since the base release.

Oracle Retail Documentation on the Oracle Technology Network

Documentation is packaged with each Oracle Retail product release. Oracle Retail product documentation is also available on the following Web site:
http://www.oracle.com/technology/documentation/oracle_retail.html

(Data Model documents are not available through Oracle Technology Network. These documents are packaged with released code, or you can obtain them through My Oracle Support.)

Documentation should be available on this Web site within a month after a product release.

Conventions

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

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

1

Introduction

Content of This Volume

This volume describes the important features that are necessary to run the Pro*C programs and the RETL programs associated with RMS. Additional RMS configuration and operations information is also included in this volume. Topics include:

- Pro*C Restart and Recovery
- Pro*C Multi-Threading
- Pro*C Array Processing
- Pro*C Input and Output Formats
- RETL Program Overview for RMS/ReSA Extractions
- RETL Program Overview for the RMS-RPAS Interface
- Internationalization
- Custom Post Processing
- Integrating RMS with Oracle Retail Workspace
- Setting up Oracle Business Intelligence Publisher

2

Pro*C Restart and Recovery

RMS has implemented a restart recovery process in most of its batch architecture. The general purpose of restart/recovery is to:

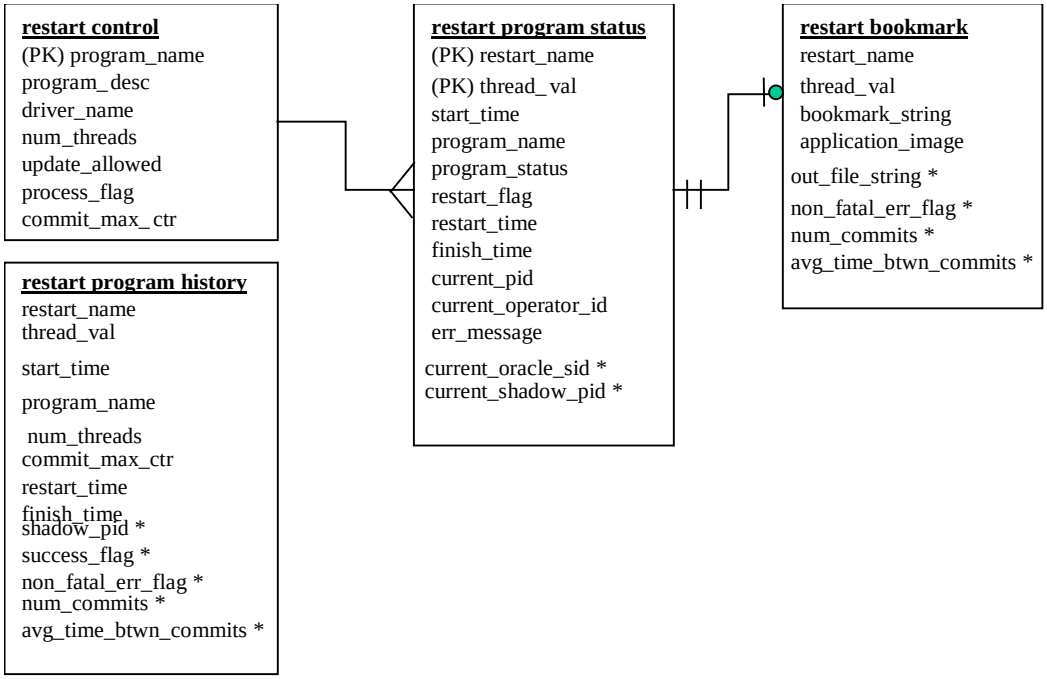
- Recover a halted process from the point of failure
- Prevent system halts due to large numbers of transactions
- Allow multiple instances of a given process to be active at the same time

Further, the RMS restart/recovery tracks batch execution statistics and does not require DBA authority to execute.

The restart capabilities revolve around a program's logical unit of work (LUW). A batch program processes transactions, and commit points are enabled based on the LUW. LUWs consist of a relatively unique transaction key (such as sku/store) and a maximum commit counter. Commit events take place after the number of processed transaction keys meets or exceeds the maximum commit counter. For example, every 10,000 sku/store combinations, a commit occurs. At the time of the commit, key data information that is necessary for restart is stored in the restart tables. In the event of a handled or un-handled exception, transactions will be rolled back to the last commit point, and upon restart the key information will be retrieved from the tables so that processing can continue from the last commit point.

Table Descriptions and Definitions

The RMS restart/recovery process is driven by a set of four tables. Refer to the Entity Relationshsips diagram below and the table descriptions that follow.



Entity Relationships

Note: The fields with asterisks (*) are only used by new batch programs of release 9.0 or later.

restart_control

The restart_control table is the master table in the restart/recovery table set. One record exists on this table for each batch program that is run with restart/recovery logic in place. The restart/recovery process uses this table to determine:

- Whether the restart/recovery is table-based or file-based
- The total number of threads used for each batch program
- The maximum records that will be processed before a commit event takes place
- The driver for the threading (multi-processing) logic.

restart_control			
(PK) program_name	varchar2	25	Batch program name
program_desc	varchar2	50	A brief description of the program function
driver_name	varchar2	25	Driver on query, for example, department (non-updatable)
num_threads	num	10	Number of threads used for current process
update_allowed	varchar2	2	Indicates whether user can update thread numbers or if done programmatically
process_flag	varchar2	1	Indicates whether process is table-based (T) or file-based (F)
commit_max_ctr	num	6	Numeric maximum value for counter before commit occurs

restart_program_status

The restart_program_status table is the table that holds record keeping information about current program processes. The number of rows for a program on the status table will be equal to its num_threads value on the restart_control table. The status table is modified during restart/recovery initialization and close logic. For table-based processing, the restart/recovery initialization logic will assign the next available thread to a program based on the program status and restart flag. For file-based processing, the thread value is passed in from the input file name. Once a thread has been assigned the program_status is updated to prevent the assignment of that thread to another process. Information will be logged on the current status of a given thread, as well as record keeping information such as operator and process timing information.

> **Setup Note:** Allow row level locking and 'dirty reads' (do not wait for rows to be unlocked for table read).

restart_program_status

(PK)restart_name	varchar2	50	Program name
(PK)thread_val	num	10	Thread counter
start_time	date		dd-mon-yy hh:mi:ss
program_name	varchar2	25	Program name
program_status	varchar2	25	Started, aborted, aborted in init, aborted in process, aborted in final, completed, ready for start
restart_flag	varchar2	1	Automatically set to 'N' after abnormal end, must be manually set to 'Y' for program to restart
restart_time	date		dd-mon-yy hh:mi:ss
finish_time	date		dd-mon-yy hh:mi:ss
current_pid	num	15	Starting program id
current_operator_id	varchar2	20	Operator that started the program
err_message	varchar2	255	Record that caused program abort & associated error message
current_oracle_sid	num	15	Oracle SID for the session associated with the current process
current_shadow_pid	num	15	O/S process ID for the shadow process associated with the current process. It is used to locate the session trace file when a process is not finished successfully.

restart_program_history

The restart_program_history table will contain one record for every successfully completed program thread with restart/recovery logic. Upon the successful completion of a program thread, its record on the restart_program_status table will be inserted into the history table. Table purgings will be at user discretion.

restart_program_history			
restart_name	varchar2	50	Program name
thread_val	Num	10	Thread counter
start_time	Date		dd-mon-yy hh:mi:ss
program_name	varchar2	25	Program name
num_threads	Num	10	Number of threads
commit_max_ctr	num	6	Numeric maximum value for counter before commit occurs
restart_time	date		dd-mon-yy hh:mi:ss
finish_time	date		dd-mon-yy hh:mi:ss
shadow_pid	num	15	O/S process ID for the shadow process associated with the process. It is used to locate the session trace file.
success_flag	varchar2	1	Indicates whether the process finished successfully (reserved for future use)
non_fatal_err_flag	varchar2	1	Indicates whether non-fatal errors have occurred for the process
num_commits	num	12	Total number of commits for the process. The possible last commit when restart/recovery is closed is not counted.
avg_time_btwn_commits	num	12	Accumulated average time between commits for the process. The possible last commit when restart/recovery is closed is not counted.

restart_bookmark

When a restart/recovery program thread is currently active, its state is started or aborted, and a record for it exists on the restart_bookmark table. Restart/recovery initialization logic inserts the record into the table for a program thread. The restart/recovery commit process updates the record with the following restart information:

- A concatenated string of key values for table processing
- A file pointer value for file processing
- Application context information such as counters and accumulators

The restart/recovery closing process will delete the program thread record if the program finishes successfully. In the event of a restart, the program thread information on this table will allow the process to begin from the last commit point.

restart_bookmark			
(PK) restart_name	varchar2	50	Program name
(PK) thread_val	num	10	Thread counter
bookmark_string	varchar2	255	Character string of key of last committed record.
application_image	varchar2	1000	application parameters from the last save point.
out_file_string	varchar2	255	Concatenated file pointers (UNIX sometimes refers to these as stream positions) of all the output files from the last commit point of the current process. It is used to return to the right restart point for all the output files during restart process.
non_fatal_err_flag	varchar2	1	Indicates whether non-fatal errors have occurred for the current process.
num_commits	num	12	Number of commits for the current process. The possible last commit when restart/recovery is closed is not counted.
avg_time_btwn_commits	num	12	Average time between commits for the current process. The possible last commit when restart/recovery is closed is not counted.

v_restart_x

Restart views will be used for query-based programs that require multi-threading. Separate views will be created for each threading driver, for example, department or store. A join will be made to a view based on threading driver to force the separation of discrete data into particular threads. Please see the threading discussion for more details.

v_restart_x		
driver_name	varchar2	- example dept, store, region, etc.
num_threads	number	total number of threads in set (defined on restart control)
driver_value	number	- will be the numeric value of the driver_name
thread_val	number	thread value defined for driver_value and num_threads combination

Restart and Recovery Data Model Design

The restart_program_status and restart_bookmark are separate tables. This is because the initialization process needs to fetch all of the rows associated with restart_name/schema, but will only update one row. The commit process will continually lock a row with a specific restart_name and thread_val. The data involved with these two processes is separated into two tables to reduce the number of hangs that could occur due to locked rows. Even if you allow 'dirty reads' on locked rows, a process will still hang if it attempts to do an update on a locked row. The commit process is only interested in a unique row, so if we move the commit process data to a separate table with row level (not page level) locking, there will not be contention issues during the commit. With the separate tables, the initialization process will now see fewer problems with contention because rows will only be locked twice, at the beginning and end of the process.

Physical Set-Up

The restart/recovery process needs to be as robust as possible in the event of database related failure. The costs outweigh the benefits of placing the restart/recovery tables in a separate database. The tables should, however, be setup in a separate, mirrored table space with a separate rollback segment.

Table and File-Based Restart/Recovery

The restart/recovery process works by storing all the data necessary to resume processing from the last commit point. Therefore, the necessary information will be updated on the restart_bookmark table before the processed data is committed. Query-based and file-based modules will store different information on the restart tables, and will therefore call different functions within the restart/recovery API to perform their tasks.

When a program's process is query-based, that is, a module is driven by a driving query that processes the retrieved rows, then the information that is stored on the restart_bookmark table is related to the data retrieved in the driving query. If the program fails while processing, the information that is stored on the restart-tables can be used in the conditional where-clause of the driving query to only retrieve data that has yet to be processed since the last commit event.

File-based processing, however, simply needs to store the file location at the time of the last commit point. This file's byte location is stored on the restart_bookmark table and will be retrieved at the time of a restart. This location information will be used to seek forward in the re-opened file to the point at which the data was last committed.

Because there is different information being saved to and retrieved from the restart_bookmark table for each of the different types of processing, different functions will need to be called to perform the restart/recovery logic. The query-based processing will call the restart_init or retek_init and restart_commit or retek_commit functions while the file-based processing will call the restart_file_init and restart_file_commit functions.

In addition to the differences in API function calls, the batch processing flow of the restart/recovery will differ between the files. Table-based restart/recovery will need to use a priming fetch logical flow, while the file-based processing will usually read lines in a batch. Table-based processing requires its structure to ensure that the LUW key has changed before a commit event can be allowed to occur, while the file-based processing does not need to evaluate the LUW, which can typically be thought of as the type of transaction being processed by the input file.

The following diagram depicts *table*-based Restart/Recovery program flow:

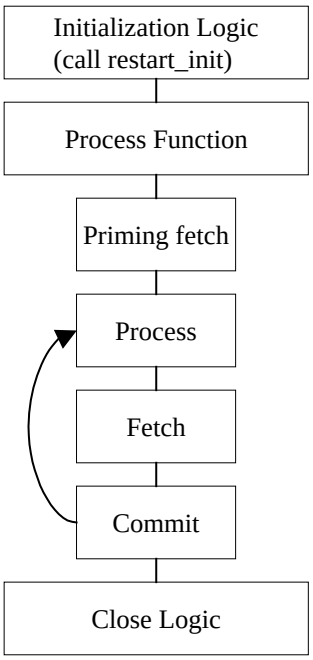
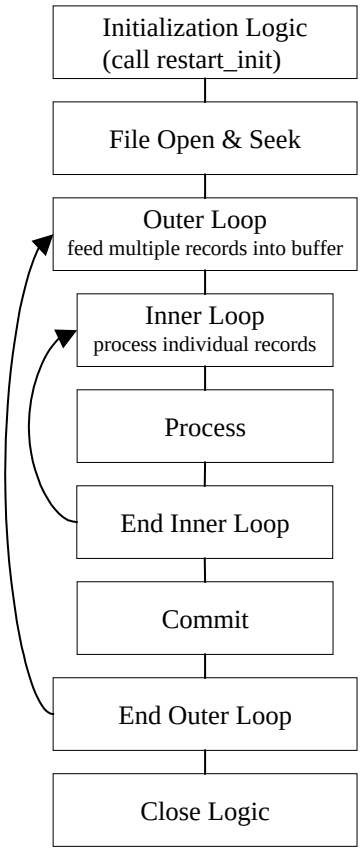


Table-Based Restart/Recovery Program Flow

The following diagram depicts *file*-based Restart/Recovery program flow



File-based Restart/Recovery Program Flow

Initialization logic:

- Variable declarations
- File initialization
- Call restart_init() or restart_file_init() function - will determine start or restart logic
- First fetch on driving query

Start logic: initialize counters/accumulators to start values

Restart logic:

- Parse application_image field on bookmark table into counters/accumulators
- Initialize counters/accumulators to values of parsed fields

Process/commit loop:

- Process updates and manipulations
- Fetch new record
- Create varchar from counters/accumulators to pass into application_image field on restart_bookmark table
- Call restart_commit() or restart_file_commit()

Close logic:

- Reset pointers
- Close files/cursors
- Call restart_close()

API Functional Descriptions

restart_init

An initialization function for table-based batch processing.

The process gathers information from the restart control tables

- Total number of threads for a program and thread value assigned to current process.
- Number of records to loop through in driving cursor before commit (LUW).
- Start string - bookmark of last commit to be used for restart or a null string if current process is an initial start and initializes the restart record-keeping (restart_program_status).
- Program status is changed to 'started' for the first available thread.
- Operational information is updated: operator, process, start_time, etc. and bookmarking (restart_bookmark) tables.
- On an initial start, a record is inserted.
- On restart, the start string and application context information from the last commit is retrieved.

restart_file_init

An initialization function for file-based batch processing. It is called from program modules.

1. The process gathers information from the restart control tables:
   - number of records to read from file for array processing and for commit cycle
   - file start point- bookmark of last commit to be used for restart or 0 for initial start
2. The process initializes the restart record-keeping (restart_program_status):
   - program status is changed to 'started' for the current thread
   - operational information is updated: operator, process, start_time, etc.
3. The process initializes the restart bookmarking (restart_bookmark) tables:
   - on an initial start, a record is inserted
   - on restart, the file starting point information and application context information from the last commit is retrieved

restart_commit

A function that commits the processed transaction for a given number of driving query fetches. It is called from program modules.

The process updates the restart_bookmark start string and application image information if a commit event has taken place:

- the current number of driving query fetches is greater than or equal to the maximum set in the restart_program_status table (and fetched in the restart_init function)
- the bookmark string of the last processed record is greater than or equal to the maximum set in the restart_program_status table (and fetched in the restart_init function)
- the bookmark string increments the counter
- the bookmark string sets the current string to be the most recently fetched key string

restart_file_commit

A function that commits processed transactions after reading a number of lines from a flat file. It is called from program modules.

The process updates the restart_bookmark table:

- start_string is set to the file pointer location in the current read of the flat file
- application image is updated with context information

restart_close

A function that updates the restart tables after program completion.

The process determines whether the program was successful. If the program finished successfully:

- the restart_program_status table is updated with finish information and the status is reset
- the corresponding record in the restart_bookmark table is deleted
- the restart_program_history table has a copy of the restart_program_status table record inserted into it
- the restart_program_status is re-initialized

If the program ends with errors

- the transactions are rolled back
- the program_status column on the restart_program_status table is set to 'aborted in *' where * is one of the three main functions in batch: init, process or final
- the changes are committed

parse_array_args

This function parses a string into components and places results into multidimensional array. It is only called within API functions and will never be called in program modules.

The process is passed a string to parse and a pointer to an array of characters.

The first character of the passed string is the delimiter.

restart_file_write

This function will append output in temporary files to final output files when a commit point is reached. It is called from program modules.

restart_cat

This function contains the logic that appends one file to another. It is only called within the restart/recovery API functions and will never be called directly in program modules.

Restart Headers and Libraries

The restart.h and the std_err.h header files are included in retek.h to utilize the restart/recovery functionality.

restart.h

This library header file contains constant, macro substitutions, and external global variable definitions as well as restart/recovery function prototypes.

The global variables that are defined include:

- the thread number assigned to the current process
- the value of the current process's thread maximum counter
  - for table-based processing, it is equal to the number of iterations of the driving query before a commit can take place
  - for file-based processing, it is equal to the number of lines that will be read from a flat file and processed using a structured array before a commit can take place
- the current count of driving query iterations used for table-based processing or the current array index used in file-based processing
- the name assigned to the program/logical unit of work by the programmer. It is the same as the restart_name column on the restart_program_status, restart_program_history, and restart_bookmark tables

std_rest.h

This library header file contains standard restart variable declarations that are used visible in program modules.

The variable definitions that are included are:

- the concatenated string value of the fetched driving query key that is currently being processed
- the concatenated string value of the fetched driving query key that is next to be processed
- the error message passed to the restart_close function and updated to restart_program_status
- concatenated string of application context information, for example, counters & accumulators
- the name of the threading driver, for example, department, store, warehouse, etc.
- the total number of threads used by this program
- the pointer to pass to initialization function to retail number of threads value

Updated Restart Headers and Libraries

Restart/recovery performs the following, among other capabilities:

- Organizes global variables associated with restart recovery
- Allows the batch developer full control of restart recovery variables parameter passing during initialization
- Removes temporary write files to speed up the commit process
- Moves more information and processing from the batch code into the library code
- Adds more information into the restart recovery tables for tuning purposes

retek_2.h

This library header file is included by all C code within Retek and serves to centralize system includes, macro defines, globals, function prototypes, and, especially, structs for use in the new restart/recovery library.

The globals used by the old restart/recovery library are all discarded. Instead, each batch program declares variables needed and calls retek_init() to get them populated from restart/recovery tables. Therefore, only the following variables are declared:

- gi_no_commit: flag for NO_COMMIT command line option (used for tuning purposes)
- gi_error_flag: fatal error flag
- gi_non_fatal_err_flag: non-fatal error flag

In addition, a rtk_file struct is defined to handle all file interfaces associated with restart/recovery. Operation functions on the file struct are also defined.

```
#define NOT_PAD           1000  /* Flag not to pad thread_val */
#define PAD               1001  /* Flag to pad thread_val at the end */
#define TEMPLATE          1002  /* Flag to pad thread_val using filename template
*/
#define MAX_FILENAME_LEN   50
typedef struct
{
   FILE* fp;                              /* File pointer */
   char  filename[MAX_FILENAME_LEN + 1];  /* Filename */
   int   pad_flag;   /* Flag whether to pad thread_val to filename */
} rtk_file;

int   set_filename(rtk_file* file_struct, char* file_name, int pad_flag);
FILE* get_FILE(rtk_file* file_struct);
int   rtk_print(rtk_file* file_struct, char* format, ...);
int   rtk_seek(rtk_file* file_struct, long offset, int whence);
```

The parameters retek_init() needs to populate are required to be passed in using a format known to retek_init(). A struct is defined here for this purpose. An array of parameters of this struct type is needed at each batch program. Other requirements are:

Need to be initialized at each batch program.

- The lengths of name, type and sub_type should not exceed the definitions here.
- Type can only be: "int", “uint”, "long", "string", or "rtk_file".
- For type "int", “uint” or "long", use "" as sub_type.
- For type "string", sub_type can only be "S" (start string) unless the string is the thread value or number of threads, in which case use “” as sub_type or "I" (image string).
- For type "rtk_file", sub_type can only be "I" (input) or "O" (output).

```
#define NULL_PARA_NAME         51
#define NULL_PARA_TYPE         21
#define NULL_PARA_SUB_TYPE     2
typedef struct
{
   char name[NULL_PARA_NAME];
   char type[NULL_PARA_TYPE];
   char sub_type[NULL_PARA_SUB_TYPE];
} init_parameter;
```

New Restart/Recovery Functions

Starting from release 9.0, all new batch programs are coded using the new restart/recovery functions. Batch programs using the old restart/recovery API functions are still in use. Therefore, Oracle Retail is currently maintaining two sets of restart/recovery libraries.

int retek_init(int num_args, init_parameter *parameter, ...)

retek_init initializes restart/recovery (for both table- and file-based):

1. Pass in num_args as the number of elements in the init_parameter array, then the init_parameter array, then variables a batch program needs to initialize in the order and types defined in the init_parameter array. Note that all int, uint and long variables need to be passes by reference.
2. Get all global and module level values from databases.
3. Initialize records for RESTART_PROGRAM_STATUS and RESTART_BOOKMARK.
4. Parse out user-specified initialization variables (variable arg list).
5. Return NO_THREAD_AVAILABLE if no qualified record in RESTART_CONTROL or RESTART_PROGRAM_STATUS.
6. Commit work.

int retek_commit(int num_args, ...)

retek_commit checks and commits if needed (for both table- and file-based):

1. Pass in num_args, then variables for start_string first, and those for image string (if needed) second. The num_args is the total number of these two groups. All are string variables and are passed in the same order as in retek_init();
2. Concatenate start_string either from passed in variables (table-based) or from ftell of input file pointers (file-based);
3. Check if commit point reached (counter check and, if table-based, start string comparison);
4. If reached, concatenated image_string from passed in variables (if needed) and call internal_commit() to get out_file_string and update RESTART_BOOKMARK;
5. If table-based, increment pl_current_count and update ps_cur_string.

int commit_point_reached(int num_args, ...)

commit_point_reached checks if the commit point has been reached (for both table- and file-based). The difference between this function and the check in retek_commit() is that here the pl_current_count and ps_cur_string are not updated. This checking function is designed to be used with retek_force_commit(), and the logic to ensure integrity of LUW exists in user batch program. It can also be used together with retek_commit() for extra processing at the time of commit.

1. Pass in num_args, then all string variables for start_string in the same order as in retek_init(). The num_args is the number of variables for start_string. If no start_string (as in file-based), pass in NULL.
2. For table-based, if pl_curren_count reaches pl_max_counter and if newly concatenated bookmark string is different from ps_cur_string, return 1; otherwise return 0.
3. For file-based, if pl_curren_count reaches pl_max_counter return 1; otherwise return 0.

int retek_force_commit(int num_args, ...)

retek_force_commit always commits (for both table- and file-based):

1. Pass in num_args, then variables for start_string first, and those for image string (if needed) second. The num_args is the total number of these two groups. All are string variables and are passed in the same order as in retek_init().
2. Concatenate start_string either from passed in variables (table-based) or from ftell of input file pointers (file-based).
3. Concatenated image_string from passed in variables (if needed) and call internal_commit() to get out_file_string and update RESTART_BOOKMARK.
4. If table-based, increment pl_current_count and update ps_cur_string.

int retek_close(void)

retek_close closes restart/recovery (for both table- and file-based):

1. If gi_error_flag or NO_COMMIT command line option is TRUE, rollback all database changes.
2. Update RESTART_PROGRAM_STATUS according to gi_error_flag.
3. If no gi_error_flag, insert record into RESTART_PROGRAM_HISTORY with information fetched from RESTART_CONTROL, RESTART_PROGRAM_BOOKMARK and RESTART_PROGRAM_STATUS tables.
4. If no gi_error_flag, delete RESTART_BOOKMARK record.
5. Commit work.
6. Close all opened file streams.

Int retek_refresh_thread(void)

Refreshes a program's thread so that it can be run again.

1. Updates the RESTART_PROGRAM_STATUS record for the current program's PROGRAM_STATUS to be 'ready for start'.
2. Deletes any RESTART_BOOKMARK records for the current program.
3. Commits work.

void increment_current_count(void)

increment_current_count increases pl_current_count by 1.

> **Note:** This is called from get_record() of intrface.pc for file-based I/O.

int parse_name_for_thread_val(char* name)

parse_name_for_thread_val parses thread value from the extension of the specified file name.

int is_new_start(void)

is_new_start checks if current run is a new start; if yes, return 1; otherwise 0.

Query-Based Commit Thresholds

The restart capabilities revolve around a program's logical unit of work (LUW). A batch program processes transactions and enables commit points based on the LUW. An LUW is comprised of a transaction key (such as item-store) and a maximum commit counter. Commit events occur after a given number of transaction keys are processed. At the time of the commit, key data information that is necessary for restart is stored in the restart table. In the event of a handled or un-handled exception, transactions will be rolled back to the last commit point. Upon restart the restart key information will be retrieved from the tables so that processing can resume with the unprocessed data.

3

Pro*C Multi-Threading

Processing multiple instances of a given program can be accomplished through "threading". This requires driving cursors to be separated into discrete segments of data to be processed by different threads. This will be accomplished through stored procedures that will separate threading mechanisms (for example, departments or stores) into particular threads given value (for example, department 1001) and the total number of threads for a given process.

File-based processing will not truly "thread" its processing. The same data file will never be acted upon by multiple processes. Multi-threading will be accomplished by dividing the data into separate files each of which will be acted upon by a separate process. The thread value is related to the input file. This is necessary to ensure that the appropriate information can be tied back to the relevant file in the event of a restart.

RMS has a store length of ten digits. Therefore, thread values, which can be based upon the store number, should allow ten digits as well. Due to the thread values being declared as 'C' variables of type int (long), the system is restricting thread values to nine digits.

This does not mean that you cannot use ten digit store numbers. It means that if you do use ten digit store numbers you cannot use them as thread values.

Threading Description

The use of multiple threads or processes in Oracle Retail batch processing will increase efficiency and decrease processing time. The design of the threading process has allowed maximum flexibility to the end user in defining the number of processes over which a program should be divided.

Originally, the threading function was going to be used directly in the driving queries. This was found, however, to be unacceptably slow. Instead of using the function call directly in the driving queries, the designs call for joining driving query tables to a view (for example, v_restart_store) that includes the function.

Threading Function for Query-Based

A stored procedure has been created to determine thread values. Restart_thread_return returns a thread value derived from a numeric driver value, such as department number, and the total number of threads in a given process. Retailers should be able to determine the best algorithm for their design, and if a different means of segmenting data is required, then either the restart_thread_return function can be altered, or a different function can be used in any of the views in which the function is contained.

Currently the restart_thread_return function is a very simple modulus routine:

```
CREATE OR REPLACE FUNCTION RESTART_THREAD_RETURN(in_unit_value  NUMBER,
                          in_total_threads NUMBER)
  RETURN NUMBER IS
  ret_val NUMBER;
BEGIN
  ret_val := MOD(ABS(in_unit_value),in_total_threads) + 1;
  RETURN ret_val;
END;
```

Restart View for Query-Based

Each restart view will have four elements:

- the name of the threading mechanism, driver_name
- the total number of threads in a grouping, num_threads
- the value of the driving mechanism, driver_value
- the thread value for that given combination of driver_name, num_threads, and driver value, thread_val

The view will be based on the restart_control table and an information table such as DEPS or STORES. A row will exist in the view for every driver value and every total number of threads value. Therefore, if a retailer were to always use the same number of threads for a given driver (dept, store, etc.), then the view would be relatively small. As an example, if all of a retailer's programs threaded by department have a total of 5 threads, then the view will contain only one value for each department. For example, if there are 10 total departments, 10 rows will exist in v_restart_dept. However, if the retailer wants to have one of the programs to have ten threads, then there will bet 2 rows for every department: one for five total threads and one for ten total threads (for example, if 10 total departments, 20 rows will exist in v_restart_dept). Obviously, retailers should be advised to to keep the number of total thread values for a thread driver to a minimum to reduce the scope of the table join of the driving cursor with the view.

Below is an example of how the same driver value can result in differing thread values. This example uses the restart_thread_return function as it currently is written to derive thread values.

Driver_name	num_threads	driver_val	thread_val
DEPT	1	101	1
DEPT	2	101	2
DEPT	3	101	3
DEPT	4	101	2
DEPT	5	101	2
DEPT	6	101	6
DEPT	7	101	4

Below is an example of what a distribution of stores might look like given 10 stores and 5 total threads:

Driver_name	num_threads	driver_val	thread_val
STORE	5	1	2
STORE	5	2	3
STORE	5	3	4
STORE	5	4	5
STORE	5	5	1
STORE	5	6	2
STORE	5	7	3
STORE	5	8	4
STORE	5	9	5
STORE	5	10	1

View syntax:

The following is an example of the syntax needed to create the view for the multi-threading join, created with script (see threading discussion for details on restart_thread_return function):

```
create or replace view v_restart_store as
 select rc.driver_name driver_name,
     rc.num_threads num_threads,
     s.store driver_value,
     restart_thread_return(s.store, rc.num_threads) thread_val
  from restart_control rc, store s
  where rc.driver_name = 'STORE'
```

There is a different threading scheme used within Oracle Retail Sales Audit (ReSA). Because ReSA needs to run 24 hours a day and seven days a week, there is no batch window. This means that there may be batch programs running at the same time that there are online users. ReSA solved this concurrency problem by creating a locking mechanism for data that is organized by store days. These locks provide a natural threading scheme. Programs that cycle through all of the store day data attempt to lock the store day first. If the lock fails, the program simply goes on to the next store day. This has the affect of automatically balancing the workload between all of the programs executing.

Thread Scheme Maintenance

All program names will be stored on the restart_control table along with a functional description, the query driver (dept, store, class, etc.) and the user-defined number of threads associated with them. Users should be able to scroll through all programs to view the name, description, and query driver, and if the update_allowed flag is set to true, to modify the number of threads (update is set to true).

File-Based

File based processing does not truly “multi-thread” and therefore the number of threads defined on restart_control will always be one. However, a restart_program_status record will need to be created for each input file that is to be processed for the program module. Further, the thread value that is assigned should be part of the input file name. The restart_parse_name function that is included in the program module will parse the thread value from the program name and use that to determine the availability and restart requirements on the restart_program_status table.

Refer to the beginning of this multi-threading section for a discussion of limits on using large (greater than nine digits) thread values.

Query-Based

When the number of threads is modified in the restart_control table, the form should first validate that no records for that program are currently being processed in the restart_program_status_table (that is, all records = 'Completed'). The program should insert or delete rows depending on whether the new thread number is greater than or less than the old thread number. In the event that the new number is less than the previous number, all records for that program_name with a thread number greater than the new thread number will be deleted. If the new number is greater than the old number, new rows will be inserted. A new record will be inserted for each restart_name/thread_val combination.

For example if the batch program SALDLY has its number of processes changed from 2 to 3, then an additional row (3) will be added to the restart_program_status table. Likewise, if the number of threads was reduced to 1 in this example, rows 2 and 3 would be deleted.

Original restart_program_status table:

row #	restart_name	thread_val	program_name etc…	
1	SALDLY	1	SALDLY	…
2	SALDLY	2	SALDLY	…

restart_program_status table after insert:

row #	restart_name	thread_val	program_name etc…	
1	SALDLY	1	SALDLY	…
2	SALDLY	2	SALDLY	…
3	SALDLY	3	SALDLY	…

restart_program_status table after delete:

row #	restart_name	thread_val	program_name etc…	
1	SALDLY	1	SALDLY	…

Users should also be able to modify the commit_max_ctr column in restart_program_status table. This will control the number of iterations in driving query or the number of lines read from a flat file that determine the logical unit of work (LUW).

Batch Maintenance

Users should be able to view the status of all records in restart_program_status table. This is where the user will come to view error messages from aborted programs, and statistics and histories of batch runs. The only fields that will be modifiable will be program_status and restart_flag. The user should be able to reset the restart_flag to 'Y' from 'N' on records with a status of aborted, started records to aborted in the event of an abend (abnormal termination), and all records in the event of a restore from tape/re-run of all batch.

Scheduling and Initialization of Restart Batch

Before any batch with restart/recovery logic is run, an initialization program should be run to update the status in the restart_program_status table. This program should update the program_status to 'ready for start' wherever a record's program_status is 'completed.' This will leave unchanged all programs that ended unsuccessfully in the last batch run.

Pre- and Post-Processing

Due to the nature of the threading algorithm, individual programs might need a pre or a post program run to initialize variables or files before any of the threads have run or to update final data once all the threads are run. The decision was made to create pre-programs and post-programs in these cases rather than let the restart/recovery logic decide whether the currently processed thread is the first thread to start or the last thread to end for a given program.

4

Pro*C Array Processing

Oracle Retail batch architecture uses array processing to improve performance wherever possible. Instead of processing SQL statements using scalar data, data is grouped into arrays and used as bind variables in SQL statements. This improves performance by reducing the server/client and network traffic.

Array processing is used for select, insert, delete, and update statements. Oracle Retail typically does not statically define the array sizes, but uses the restart maximum commit variable as a sizing multiple. Users should keep this in mind when defining the system's maximum commit counters.

An important factor to keep in mind when using array processing is that Oracle does not allow a single array operation to be performed for more than 32000 records in one step. The Oracle Retail restart/recovery libraries have been updated to define macros for this value: MAX_ORACLE_ARRAY_SIZE.

All batch programs that use array processing need to limit the size of their array operations to MAX_ORACLE_ARRAY_SIZE.

If the commit max counter is used for array processing size, check it after the call to restart_init() and, if necessary, reset it to the maximum value if greater. If retek_init() is used to initialize, check the returned commit max counter and reset it to the maximum size if it is greater. In case of retek_init(), reset the library's internal commit max counter by calling extern int limit_commit_max_ctr(unsigned int new_max_ctr).

If some other variable is used for sizing the array processing, the actual array-processing step will have to be encapsulated in a calling loop that performs the array operation in sub segments of the total array size where each sub-segment is at most MAX_ORACLE_ARRAY_SIZE large. Currently all Oracle Retail batch programs are implemented this way.

5

Pro*C Input and Output Formats

Oracle Retail batch processing will utilize input from both tables and flat files. Further, the outcome of processing can both modify data structures and write output data. Interfacing Oracle Retail with external systems is the main use of file based I/O.

General Interface Discussion

To simplify the interface requirements, Oracle Retail requires that all in-bound and out-bound file-based transactions adhere to standard file layouts. There are two types of file layouts, detail-only and master-detail, which are described below.

An interfacing API exists within Oracle Retail to simplify the coding and the maintenance of input files. The API provides functionality to read input from files, ensure file layout integrity, and write and maintain files for rejected transactions.

Standard File Layouts

The RMS interface library supports two standard file layouts; one for master/detail processing, and one for processing detail records only. True sub-details are not supported within the RMS base package interface library functions.

A 5-character identification code or record type identifies all records within an I/O file, regardless of file type. Valid record type values include the following:

- FHEAD—File Header
- FDETL—File Detail
- FTAIL—File Tail
- THEAD—Transaction Header
- TDETL—Transaction Detail
- TTAIL—Transaction Tail

Each line of the file must begin with the record type code followed by a 10-character record ID.

Detail-Only Files

File layouts have a standard file header record, a detail record for each transaction to be processed, and a file trailer record. Valid record types are FHEAD, FDETL, and FTAIL.

Example:

```
FHEAD0000000001STKU1996010100000019960929
FDETL0000000002SKU100000040000011011
FDETL0000000003SKU100000050003002001
FDETL0000000004SKU100000050003002001
FTAIL0000000050000000003
```

Master and Detail Files

File layouts will have a standard file header record, a set of records for each transaction to be processed, and a file trailer record. The transaction set will consist of a transaction set header record, a transaction set detail for detail within the transaction, and a transaction trailer record. Valid record types are FHEAD, THEAD, TDETL, TTAIL, and FTAIL.

Example:

```
FHEAD0000000001RTV 19960908172000
THEAD000000000200000000000000119960909120200000000003R
TDETL000000000300000000000001000001SKU10000012
TTAIL0000000004000001
THEAD000000000500000000000002199609091202001215720131R
TDETL000000000600000000000002000001UPC400100002667
TDETL000000000700000000000020000021UPC400100002643 0
TTAIL0000000008000002
FTAIL0000000090000000007
```

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	Specified by external system	Line number of the current file
	File Type Definition	Char(4)	n/a	Identifies transaction type
	File Create Date	Date	Create date	Date file was written by external system
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type
	File Line Identifier	Number(10)	Specified by external system	Line number of the current file
	Transaction Set Control Number	Char(14)	Specified by external system	Used to force unique transaction check
	Transaction Date	Char(14)	Specified by external system	Date the transaction was created in external system
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Identifier	Number(10)	Specified by external system	Line number of the current file
	Transaction Set Control Number	Char(14)	Specified by external system	Used to force unique transaction check

Record Name	Field Name	Field Type	Default Value	Description
	Detail Sequence Number	Char(6)	Specified by external system	Sequential number assigned to detail records within a transaction
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Identifier	Number(10)	Specified by external system	Line number of the current file
	Transaction Detail Line Count	Number(6)	Sum of detail lines	Sum of the detail lines within a transaction
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	Specified by external system	Line number of the current file
	Total Transaction Line Count	Number(10)	Sum of all transaction lines	All lines in file less the file header and trailer records

Electronic Data Interchange (EDI)

Starting with release 7.0, EDI files used or created by RMS are in a generic format: RMS no longer supports particular EDI standards. By processing EDI output and input in a generic format, RMS is no longer limited to a single standard, which allows Oracle Retail customers to better utilize any and all standards they choose to use. Translating EDI input and output files into any format from any format by third-party software is an industry "best practice".

Formerly, EDI transactions in RMS conformed to ASC X12/VICS (version 3040) and ANA/TRADACOMS standards. EDI transactions are now expected to be in a format that adheres to the RMS file interfacing standards. Both in-bound and out-bound files are written in a fixed field layout with standard file header and trailer records. Transaction information is included in master/detail or detail-only records. The layouts are consistent with interface files used elsewhere in the RMS.

RMS EDI batch processes write out-bound transaction files into the generic layout format, which are then translated by the third-party software into the standard required by each trading partner. The post-translated versions are transmitted to the trading partner. In-bound transactions should be formatted by the trading partner in a predetermined standard, transmitted, and then translated by the Oracle Retail retailer's translation software into the generic file layout. The generic file is used as the input file for RMS EDI batch processing.

It is impractical for Oracle Retail to continue to maintain code that supports any particular EDI standard. There are multiple viable standards that are utilized by vendors and retailers. Further, those standards have multiple versions. Most retailers are already using software to map and translate EDI transactions into the required standard or version. There are excellent third-party software packages, such as Sterling Software's Gentran™ translator, that effectively translate in-bound and out-bound transactions into the necessary formats. The use of third-party translation software is not only the common business practice, but also the best business practice of today's retailer.

6

RETL Program Overview for the RMS-RPAS Interface

Oracle Retail ETL Architecture

RMS works in conjunction with the Oracle Retail Extract Transform and Load (RETL) framework. This architecture utilizes a high performance data processing tool that allows database batch processes to take advantage of parallel processing capabilities.

The RETL framework runs and parses through the valid operators composed in XML scripts.

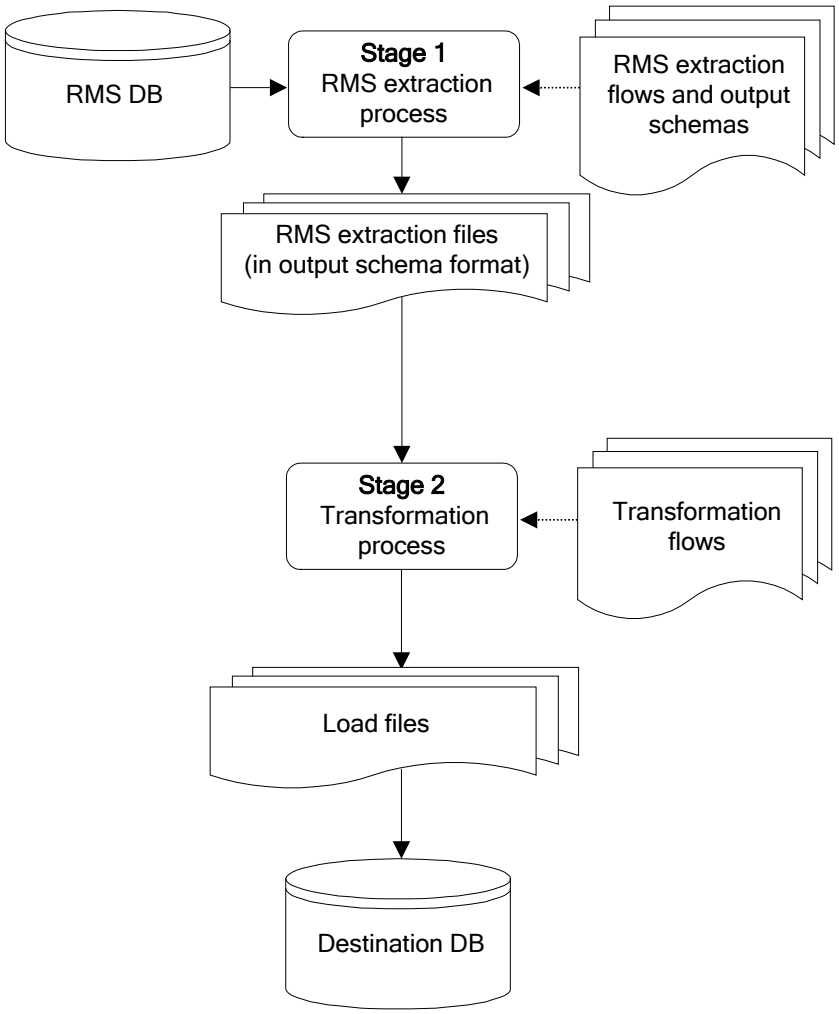
More information about the RETL tool is available in the latest RETL Programmer's Guide.

The diagram below illustrates the extraction processing architecture. Instead of managing the change captures as they occur in the source system during the day, the process involves extracting the current data from the source system. The extracted data is output to flat files. These flat files are then available for consumption by products such as RPAS.

The target system has its own way of completing the transformations and loading the necessary data into its system, where it can be used for further processing in the environment. See RPAS documentation for information related to transformations and loadings.

The architecture relies upon two distinct stages, shown in the diagram below. Stage 1 is the extraction from the RMS database using well-defined flows specific to the RMS database. The resulting output is comprised of data files written in a well-defined schema file format. This stage includes no destination specific code.

Stage 2 introduces a flow specific to the destination. In this case, flows for RPAS are designed to transform the data so that RPAS can import the data properly.



The two stages of RETL processing

RETL Program Overview

This section summarizes the RETL program features utilized in the RMS extractions and loads. Installation information about the RETL tool is available in the latest RETL Programmer's Guide.

Configuration

Version of RETL

Before trying to configure and run RMS RETL, install RETL version 12.0 or later, which is required to run RMS RETL. See the latest RETL Programmer's Guide for thorough installation information.

RETL User and Permissions

The permissions are set up as per the RETL Programmer's Guide. RMS RETL reads and writes data files and creates, deletes, updates and inserts into 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 RDF_HOME to your base directory for RMS RETL. This is the top level directory that you selected during the installation process. In .profile, you should add a line such as the following:

```
export RDF_HOME=<base directory for RMS RETL>
```

rmse_rpas_config.env Settings for RPAS

There are several constants that must be set in rmse_rpas_config.env depending upon a retailer's preferences and the local environment. These are summarized in the following table.

Constant Name	Default Value	Alternate Value	Description
DATE_TYPE	vdate	current_date	Determines whether the date used in naming the error, log, and status files is the current date or the VDATE value found in the PERIOD table.
DBNAME	rtkdev01	Depends on installation	The database schema name.
RMS_OWNER	RPASINT	Depends on installation	The username of the RMS database schema owner.
BA_OWNER		Depends on installation	The username of the RMS batch user (not currently used by RMS-RPAS).
CONN_TYPE	thin	oci	The way in which RMS connects to the database.
DBHOST	mspdev17	Depends on installation	The computer hardware node name.
DBPORT	1524	Depends on installation	The port on which the database listener resides.
LOC_ATTRIBUTES_ ACTIVE	False	True	Determines whether rmse_rpas_attributes.ksh is run or not.
PROD_ ATTRIBUTES_ACTIVE	False	True	Determines whether rmse_rpas_attributes.ksh is run or not.
DIFFS_ACTIVE	True	False	Determines whether rmse_rpas_merchhier.ksh generates data files that contain diff allocation information.

Constant Name	Default Value	Alternate Value	Description
ISSUES_ACTIVE	True	False	If set to 'True', rmse_rpas_stock_on_hand also extracts stock at the warehouse level. If set to 'False', rmse_rpas_stock_on_hand extracts stock at the store level only.
LOAD_TYPE	CONVENTIONAL	DIRECT	Data loading method to be used by SQL*Loader (Direct may be faster than conventional.)
DB_ENV	ORA	DB2, TERA	Database type (Additional changes to the software may be needed if a database other than Oracle is selected.)
NO_OF_CPUS	4	Depends on installation	Used in parallel database query hints to improve performance.
LANGUAGE	en	Various	En = English
RFX_OPTIONS	-c $RDF_HOME/rfx/etc/rfx.conf -s SCHEMAFILE	-c $RDF_HOME/rfx/etc/rfx.conf	Processing speed may be increased for some extractions if the -s SCHEMAFILE option is omitted

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

```
export PASSWORD=mypasswd
```

Be sure to review the environmental parameters in the rmse_rpas_config.env 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 constants from the table above in the rmse_rpas_config.env script as needed.

Program Return Code

RETL programs use a 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_rpas_config.env is run. This script checks 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 name and directory of the program status control file is set in the configuration script (rmse_rpas_config.env). The directory defaults to $RDF_HOME/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 the rmse_rpas_daily_sales.ksh program would be named as follows for a batch run on the business virtual date of January 5, 2001:

```
$RDF_HOME/error/rmse_rpas_daily_sales.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 module imports data from flat files, performs transformations if necessary, and then loads the data into the applicable RMS table.

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. No RMS to RPAS extraction programs have any restart/recovery capability. The single RMS load program, rmsl_rpas_forecast.ksh, takes a text file as its input, and the following two choices are available that enable the program to complete the load in the event of an error:

- Re-run the program with the entire input file.
- Re-run the program with only the input records that were not processed successfully the first time.

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. In some cases, progress messages are also written. The name and directory of the daily log file is set in the configuration script (rmse_rpas_config.env). The directory defaults to $RDF_HOME/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:

```
$RDF_HOME/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. For example:

```
rmse_rpas_item_retail 17:09:07: Program started ...
rmse_rpas_item_retail 17:09:12: Program completed successfully
```

Some error messages are also written to the log file, such as '`No output file specified`'.

Program Error File

In addition to the daily log file, each program also writes its own detailed 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.

If a program finishes unsuccessfully, a message is usually written in the error file that indicates where the problem occurred in the process.

The name and directory of the program error file is set in the applicable configuration file (rmse_rpas_config.env). The directory defaults to $RDF_HOME/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 produced during execution of the program).

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 detailed log information for the `rms_item_master.ksh` program would be placed in the following file for the batch run on the business virtual date of January 5, 2001:

```
$MMHOME/error/rms_item_master.20010105
```

RMSE Reject Files

RMSE extract 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 consist of the rejected records.

The name and directory of the reject file are defined in the applicable configuration script (rmse_rpas_config.env). The directory defaults to $RDF_HOME/data.

> **Note:** A directory specific to reject files can be created. The rmse_rpas_config.env script would need to be changed to define the reject directory constant such that it would 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

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 because they do not change on a day-to-day basis. All schema files end with ".schema" and are placed in the "$RDF_HOME/rfx/schema" directory.

Command Line Parameters

The only programs or scripts that allow command line parameters (or arguments) are the rmse_rpas_config.env script and the pre_rmse_rpas.ksh and rmse_rpas.ksh programs. All of the command line parameters for these modules are optional and are described below (the square brackets indicate that the parameter is optional):

rmse_rpas_config.env

Usage: $RDF_HOME/rfx/etc/rmse_rpas_config.env [-t $*] [-r $*] [-s $*] [-v $* | -c $*]

Description of Command Line Options

> **Note:** See the end of this description for an explanation of the need for the '$*' that appears after each command line option.

- `-t`: This option causes rmse_rpas_config.env to skip the initializing of the environment variables that obtain their values from the '.txt' files, except for VDATE

which is initialized with the date found in the vdate.txt file. This option is utilized by pre_rmse_rpas.ksh, rmse_rpas.ksh, rdft.ksh and outage.ksh when they call rmse_rpas_config.env.

- `-r`: This option prevents the redirection of all output (stdout and stderr) to the error file. This can be useful during debugging and maintenance. This option can also be utilized by rmse_rpas.ksh, rdft.ksh and outage.ksh when they call rmse_rpas_config.env.
- The '`-t`' and '`-r`' options must be followed by '`$*`' on the line which invokes this script. This step is necessary in order to preserve the command line arguments or options that may have been present on the command line for the RETL script that invokes this script. However, the '`$*`' should only appear once if both options are used.
- `-s`: This option causes rmse_rpas_config.env to skip the STATUS_FILE test. This is also useful during maintenance and debugging.
- `-v`: If DATE_TYPE (in rmse_rpas_config.env) is set to 'vdate', this option prevents the normal exit with an error message when the vdate.txt file is empty or non-existent; instead, it will use the current date to derive FILE_DATE. However, if DATE_TYPE is set to '`vdate`', and vdate.txt actually does exist and is non-empty, the date in vdate.txt continues to be used even if this option is set. If DATE_TYPE is set to '`current_date`', this option has no effect.
- `-c`: This option overrides the DATE_TYPE switch setting and causes the current date to be used to derive FILE_DATE regardless of what DATE_TYPE is set to. This option is utilized by pre_rmse_rpas.ksh when it calls rmse_rpas_config.env, if it is run with the `-c` option on its command line. The '`-c`' option is normally only used when rmse_rpas_config.env is called from pre_rmse_rpas.ksh.
- If only one command line option is used, it must be followed by '`$*`'. But if more than one option is specified, then '`$*`' must be entered on the command line only once after all options have been entered. The '`$*`' is necessary in order to preserve the command line arguments or options (if there are any) that are present on the command line that is used to execute the RETL script which invokes this script.
- If more than one option is specified, options must appear on the command line in the same order as shown on the "Usage" line, above.

pre_rmse_rpas.ksh

- Usage: pre_rmse_rpas.ksh [`-c`]
- The '`-c`' option is used to specify what option is to be placed on the rmse_rpas_config.env command line when it is called by this program. It is usually used the first time that pre_rmse_rpas.ksh is run at a new installation or if the state of the vdate.txt file is unknown. This option is passed directly to rmse_rpas_config.env when it is called by pre_rmse_rpas.ksh. No other use is made of this parameter by pre_rmse_rpas.ksh.
- This option causes rmse_rpas_config.env to use the current date to initialize FILE_DATE instead of possibly setting it to VDATE, which is obtained from the vdate.txt file. (FILE_DATE is the date that is used to name the error, log, and status files.)
- The current date is used regardless of how DATE_TYPE is set in rmse_rpas_config.env. By using the '`-c`' option, there is no need to manually set up the vdate.txt file before running this script.
- The normal mode for pre_rmse_rpas.ksh (without the `-c` option) is that when it calls rmse_rpas_config.env, FILE_DATE is set to VDATE or the current date, depending on how DATE_TYPE is set in rmse_rpas_config.env. If DATE_TYPE is set to 'vdate',

and if the vdate.txt file does not exist or is empty, rmse_rpas_config.env (and this program) exits with an error message.

- The use of this option does not affect what date is used by any of the other RETL scripts that run after this script is done. After pre_rmse_rpas.ksh has run, when the other RETL scripts are run, they call rmse_rpas_config.env with no options on the command line, and their files are named using VDATE or the current date, depending on how DATE_TYPE is set in rmse_rpas_config.env.

rmse_rpas.ksh:

- Usage: rmse_rpas.ksh `[-c]`
- The presence of the '`-c`' option causes FILE_DATE in rmse_rpas_config.env to be set to the current date instead of possibly using VDATE (which gets its value from the vdate.txt file), but only when it is called by rmse_rpas.ksh and pre_rmse_rpas.ksh (pre_rmse_rpas.ksh is invoked by rmse_rpas.ksh). It has no effect when other extract programs call rmse_rpas_config.env, at the time that they are invoked by rmse_rpas.ksh. This option is passed directly to rmse_rpas_config.env and pre_rmse_rpas.ksh when they are called by rmse_rpas.ksh. No other use is made of this parameter by rmse_rpas.ksh.

RMSE I/O File Names

Most of the output path/filenames have the format, $DATA_DIR/(RMSE_RPAS_program name).dat. Similarly, the schema format for the records in these files are specified in the file - $SCHEMA_DIR/(RMSE_RPAS_program name).schema.

Typical Run and Debugging Situations

The following examples illustrate typical run and debugging situations for programs. The log, error, etc. 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_rpas_stores.ksh:

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

```
$rmse_rpas_stores.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_rpas_stores.
2. **Data:** The rmse_rpas_stores.dat file exists in the data directory and contains the extracted records.
3. **Schema:** The rmse_rpas_stores.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_rpas_stores.20010309, contains the standard RETL flow (ending with "All threads complete" and "Flow ran successfully") and no error messages.
5. **Program status control:** The program status control file, rmse_rpas_stores.status.20010309, will not exist.
6. **Reject file:** The reject file, rmse_rpas_stores.rej.20010309, will not exist.

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_rpas_stores.
2. **Data:** The rmse_rpas_stores.dat file may exist in the data directory but may not contain all the extracted records.
3. **Schema:** The rmse_rpas_stores.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_rpas_stores.20010309, may contain one or more error messages.
5. **Program status control:** The program status control file, rmse_rpas_stores.status.20010309, exists.
6. **Reject file:** The reject file, rmse_rpas_stores.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 $RDF_HOME/rfx/src. At a UNIX prompt, enter:

```
$rmse_rpas_stores.ksh
```

RPAS/AIP Configuration

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

This section 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:

- **Log file:** Today's log file, 20010309.log, contains the messages "Program started …" and "Program completed successfully" for rmse_aip_store.
- **Data:** The rmse_aip_store.dat file exists in the data directory and contains the extracted records.
- **Schema:** The rmse_aip_store.schema file exists in the schema directory and contains the definition of the data file in #2 above.
- **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.
- **Program status control:** The program status control file, rmse_aip_store.status.20010309, does not exist.
- **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:
- **Log file:** Today's log file, 20010309.log, does not contain the "Program completed successfully" message for rmse_stores.
- **Data:** The rmse_aip_store.dat file may exist in the data directory but may not contain all the extracted records.
- **Schema:** The rmse_aip_store.schema file exists in the schema directory and contains the definition of the data file in #2 above.
- **Error file:** The program's error file, rmse_aip_store.20010309, may contain an error message.
- **Program status control:** The program status control file, rmse_aip_store.status.20010309, exists.
- **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
```

7

Internationalization

Internationalization is the process of creating software that is able to be translated more easily. Changes to the code are not specific to any particular market. RMS has been internationalized to support multiple languages.

This section describes configuration settings and features of the software that ensure that the base application can handle multiple languages.

Translation

Translation is the process of interpreting and adapting text from one language into another. Although the code itself is not translated, components of the application that are translated may include the following:

- Graphical user interface (GUI)
- Error messages
- Reports

The following components are not translated:

- Documentation (online help, release notes, installation guide, user guide, operations guide)
- Batch programs and messages
- Log files
- Configuration tools
- Demonstration data
- Training materials

The user interface for RMS has been translated into:

- Chinese (simplified)
- Chinese (traditional)
- Croatian
- Dutch
- French
- German
- Greek
- Hungarian
- Italian
- Japanese
- Korean
- Polish
- Portuguese (Brazilian)
- Russian
- Spanish
- Swedish
- Turkish

RMS User Interface Language Display Settings

The user's preferred language can be selected through the RMS UI or via the database on the user_attrib table. The user's language is connected to the user's ID in RMS and displays the translated strings in the user's selected language. RMS has a fail/safe mechanism built into the code. If the user's preferred language is not found, then RMS rolls back to English language display of the UI label.

Note: A retailer has the two options below regarding internationalization when installing the application. See the *RMS Installation Guide* for the procedures related to each.

- English and multiple secondary languages
- Install English first and then update with a translated language (fully translated non-English installation). No secondary languages are installed when your primary language is one other than English.

Multiple Languages in one RMS Forms Session

RMS allows for multiple application servers to point to a single instance of forms and database. This is specific to users who want to have a multi-lingual install of RMS and have two URLs, each with a separate NLS_LANG session setting. This helps to facilitate secondary user language sessions as some UI elements are displayed in the language of the NLS_LANG session of the application server

To set up multiple URLs:

1. Copy URL information in formsweb.cfg.
   a. Change the URL (the part in square brackets) to something new.
   b. Change the env file reference to a new file (to be created in step 2).
2. Copy the current env file to the new name created in step 1.
3. Modify the new env file by setting NLS_LANG to the new value.

Key RMS Tables Related to Internationalization

Several tables handle displayable text that can also be translated.

If the retailer creates a new form, a new menu, or a new object on a form, then the retailer will need to populate these tables with the corresponding information. If the retailer customizes the information in any of the tables FORM_ELEMENTS, FORM_ELEMENTS_LANGS, MENU_ELEMENTS, or MENU_ELEMENTS_LANGS, the base_ind field in customized records must contain 'N'. Any record with BASE_IND=N will be preserved in a temp table during future patches.

FORM_ELEMENTS

This table is used for screen display and holds the master list of items for all forms whose labels/prompts are translated. This information will always be in English. The BASE_IND=Y means that the item is part of the base Oracle Retail code set. BASE_IND =N indicates that the item was added as part of retailer customization. Anything with the BASE_IND =N will be preserved at upgrade time on the FORM_ELEMENTS_TEMP, but the retailer is responsible for moving the data back to FORM_ELEMENTS.

FORM_ELEMENTS_LANGS

This table is used for screen display. This table holds translated values for labels/prompts on forms. This information will be in a language that is defined on the lang column of the user_attrib table. All users see data from this table, as the retailer may customize the text of a given field. The access key for a button is defined by filling in the LANG_ACCESS_KEY/LANG_LABEL_PROMPT field. At run time, that character will be marked in the string, and function as the access key. Any time the retailer changes the LANG_ACCESS_KEY/LANG_LABEL_PROMPT or LANG_ACCESS_KEY/LANG_LABEL_PROMPT, the BASE_IND should be updated to N because it is not part of the base language translations provided by Oracle Retail. Anything with the BASE_IND=N will be preserved at upgrade time on the FORM_ELEMENTS_LANGS_TEMP, but the retailer is responsible for moving the data back to FORM_ELEMENTS_LANGS.

MENU_ELEMENTS

This table is used for screen display. This table holds the master list for all menus whose items are translated. This information will always be in English. The access key for a menu option is defined by using the ampersand (&) before the character that is the access key in the default description. The BASE_IND=Y means that the item is part of the base Oracle Retail code set. BASE_IND=N indicates that the item was added as part of retailer customization. Anything with the BASE_IND=N will be preserved at upgrade time on the MENU_ELEMENTS_TEMP, but the retailer is responsible for moving the data back to MENU_ELEMENTS.

MENU_ELEMENTS_LANGS

This table is used for screen display. This table holds the values for all menus whose items are translated. This information will be in a language that is defined on the lang table. Even English language users see data from this table, as the retailer may customize the text of a given menu option. Any time the retailer changes the LANG_LABEL, the BASE_IND should be updated to N because it is not part of the base language translations provided by Oracle Retail. Anything with the BASE_IND=N will be preserved at upgrade time on the MENU_ELEMENTS_LANGS_TEMP, but the retailer is responsible form moving the data back to MENU_ELEMENTS_LANGS.

FORM_MENU_LINK

This table is used for screen display. This table holds the intersection of form and menu files, mapping each form to the menu that it displays.

CODE_DETAIL_TRANS

This table holds non-primary language descriptions of code types defined on the CODE_DETAIL table. The retailer has a multi-language option.

8

Custom Post Processing

RMS has an optional method of handling unwanded cartons for customer post processing. This only applies to stock order receiving. An unwanded carton occurs when a carton was not scanned when the stock order was shipped, but is scanned at the time of the receipt. These cartons do not contain any shipment records in RMS.

Since the carton contains items that did not go through the appropriate transfer out procedure, the inventory for those items will not be accurate. As a result, the message which contains the unwanded (unscanned) carton is rejected by RMS to the RIB error hospital at the time of receiving. RMS will then publish to the warehouse management system via the RIB of the unwanded cartons in the RcptAdjustDesc message. The warehouse management system will then send RMS a shipment message containing the appropriate BOL and the carton ID. RMS will process the message and create or update the shipment records. The next time RMS tries to process the rejected receipt message with the unwanded carton, RMS will be able to process it.

The client's warehouse management system must be able to support the processing of the RcptAdjustDesc message above in order for this functionality of unwanded carton to work successfully.

9

Integrating RMS with Oracle Retail Workspace

Note: Prior to integrating RMS with Oracle Retail Workspace, it is recommended that you set up RMS for single sign-on. See the Single Sign-on appendix in the *Oracle Retail Merchandising Installation Guide* for additional details.

The Oracle Retail Workspace installer prompts you to enter the URL for your supported Oracle Retail applications. However, if a client installs a new application after Oracle Retail Workspace is installed, the retail-workspace-page-config.xml file needs to be edited to reflect the new application.

The file as supplied comes with all appropriate products configured, but the configurations of non-installed products have been "turned off". Therefore, when "turning on" a product, locate the appropriate entry, set "rendered" to "true", and enter the correct URL and parameters for the new application.

The entry consists of the main URL string plus one parameter named "config". The value of the config parameter is inserted by the installer. Somewhere in the installer property files there is a value for the properties "deploy.retail.product.rms.url" and "deploy.retail.product.rms.config".

For example, suppose RMS was installed on mycomputer.mycompany.com, port 7777, using a standard install and rms configured with the application name of "rms121sedevhpsso". If you were to access RMS directly from your browser, you would type in:

```
http://mycomputer.mycompany.com:7777/forms/frmservlet?config=rms121sedevhpsso
```

The entry in the retail-workspace-page-config.xml after installation would resemble the following:

```
<url>http://mycomputer.mycompany.com:7777/forms/frmservlet</url>
    <parameters>
        <parameter name="config">
            <value>rms121sedevhpsso</value>
        </parameter>
    </parameters>
```

10

RMS – Oracle E-Business Suite Financials Integration Using Oracle Application Integration Architecture

This chapter describes the integration between Oracle Retail systems and Oracle E-Business Suite Financials (including Oracle General Ledger and Oracle Payables), as developed and supported by Oracle Application Integration Architecture (AIA).

When the option to integrate is chosen, selected information is shared among the systems. Integration and validation services are in place to ensure the shared data matches.

> **Note:** This chapter addresses the points within Oracle Retail systems that are essential to integration. For more information about the entire integration process, including mapping to Oracle E-Business Suite data and settings, see the AIA document, *Oracle Financial Operations Control Integration Pack for OracleRetail Merchandising Suite and Oracle E-Business Suite Financials 2.5 - Implementation Guide.* For more information about Web services, see the following chapters in the *Oracle Retail Merchandising System Operations Guide, Volume 2*: "Service Provider Implementations API Designs" and "Web Services."

Participating Applications

The following Oracle Retail applications are included in the integration covered by this chapter:

- Oracle Retail Merchandising System (RMS)
- Oracle Retail Sales Audit (ReSA)
- Oracle Retail Invoice Matching (ReIM)
- Oracle Retail Integration Bus (RIB)

Assumptions and Dependencies

- The option to integrate should be selected during initial setup of the RMS system.
- ReIM accesses RMS to determine if integration is active. Initial setup of RMS must occur prior to the integration of ReIM.
- The URLs for the AIA Web services that are a necessary for this integration must be maintained in the RMS_RETAIL_SERVICE_REPORT_URL table and in the ReIM integration.properties file.
- Real time account validation is done only when the financial integration with Oracle E-Business Suite is ON.
- Partners must be set up as suppliers in Oracle E-Business Suite. Then the partner must be manually set up in RMS using the RMS Supplier ID that was generated when the Oracle E-Business Suite supplier was interfaced to Oracle Retail. Partner

functionality within RMS and ReIM can then proceed normally. The RMS supplier generated as part of this process is not used.

- Payment terms and freight terms are manually maintained.

Data Setup

Integration of Oracle Retail applications and Oracle E-Business Suite Financials relies on synchronization of essential data, such as currency exchange rates and suppliers. Through careful discussions, users of both systems determine the common codes and descriptions that will best serve their business needs.

Once agreement is reached, this information is set up and maintained. Depending on volume, some shared information is set up in Oracle Retail applications or in Oracle E-Business Suite and electronically transferred to the other system. Otherwise, shared information is set up manually within each system, and users of both systems must ensure that codes and descriptions match.

RMS Data Setup and Configuration

This section describes setup considerations for RMS data.

RMS System Options

As part of the RMS system options setup, set the following options as indicated:

- FINANCIAL_IND=Y

This system_option indicates that the Oracle Retail system is integrated with a financial system:

- FINANCIAL_AP=A

A value of A indicates that the financial system to which RMS is interfaced is Oracle E-Business Suite through Oracle Application Integration Architecture (AIA).

- GL_ROLL_UP can be D/S/C
- MULTIPLE_SET_OF_BOOKS_IND = Y
- SUPPLIER_SITE_IND = Y
- ORG_UNIT_IND = Y

Organization Units

Use the Organizational Unit window (RMS Start Menu > Control > Setup > Org Unit >Edit) to define organizational units in RMS that match those being set up in Oracle E-Business Suite. When an organizational unit is entered in RMS, the valid organizational units are those associated with the set of books (SOB) that is being used for the general ledger interface.

Currency Exchange Rates

Currency exchange rate is used to translate the monetary value of one currency in terms of another. Depending on business needs, a Currency Exchange Rate Type of Operational or Consolidation is selected for use in all transactions.

This value is set up manually in RMS and mapped to Oracle E-Business Suite through the Currency Exchange Type mapping window. Currency Exchange Rate data is owned by Oracle General Ledger, and updates are sent to Oracle Retail applications.

Determine the Exchange Type being sent by Oracle General Ledger (for example, Consolidation or Operational) that you want RMS to use. Update the

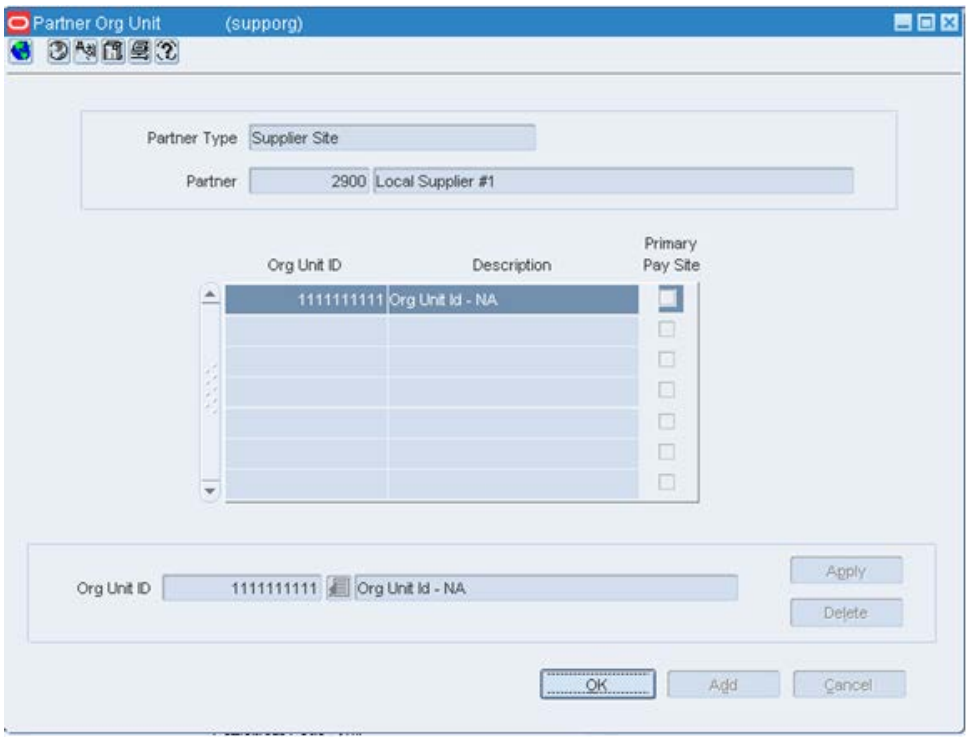
FIF_CURRENCY_XREF for mapping the external exchange type being sent by Oracle General Ledger with RMS Exchange Type.

For example, for Consolidation and Operational exchange types, the FIF_CURRENCY_XREF table holds the following entries:

FIF_EXCHANGE_TYPE	RMS_EXCHANGE_TYPE
C	C
O	O

Supplier Address Types

Within RMS, supplier information (such as Order From and Remit To addresses) is used in generating purchase orders. Oracle Payables uses supplier information for payment generation. It is important, then, that this information is synchronized.



Suppliers are created in Oracle Payables and exported to RMS. When FINANCIAL_AP is set to A, suppliers cannot be created using the RMS forms. However, after the supplier exists in RMS, all data values for the supplier (except supplier name and status) continue to be updated using the RMS forms. The association of supplier sites to organization units is accessed only in view mode through RMS forms. One supplier site per supplier organization unit combination can be marked as primary payment site.

Where SYSTEM_OPTIONS.FINANCIAL_AP is A, disable auto generate supplier/partner numbers and associated check boxes.

> **Note:** Supplier information is created, updated and inactivated only in Oracle Payables. This information is transferred from Oracle Payables to the participating Oracle Retail applications, where additional retail-specific attributes may be maintained.

Country Codes

When country codes are defined and seeded in RMS, ensure that country codes are mapped to Oracle E-Business Suite country codes through AIA DVM mapping. The following is an example of AIA DVM Mapping for COUNTRY_CODE:

EBIZ_01	COMMON	RETL_01
USA	700	US
CAN	701	CA

Financial Calendar

The financial calendar within Oracle Retail systems is manually set up and maintained separately from the Oracle General Ledger financial calendar.

Freight Terms

A freight term is an agreement between the retailer and a supplier regarding transportation charges for goods delivered by the supplier. Freight terms are used by RMS as purchase orders are generated.

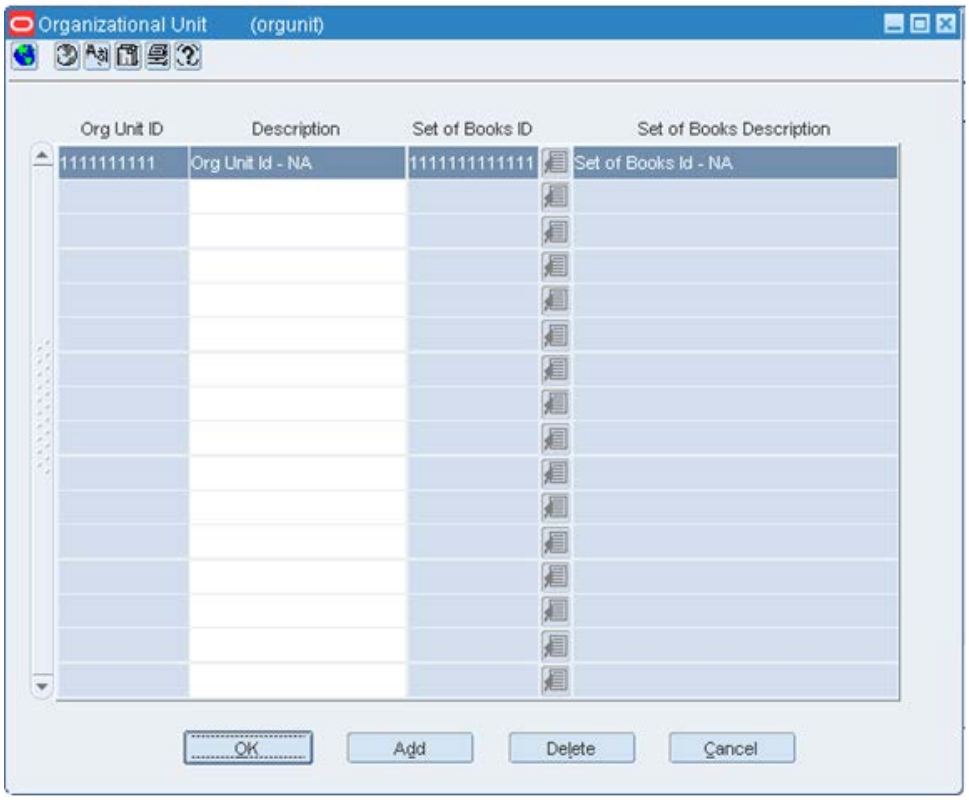
Within the RMS system, freight terms are set up and maintained manually. They also are maintained in Oracle Payables.

Payment Terms and Currency Exchange Rates

Currency exchange rates are created and updated in Oracle General Ledger and exported to RMS. Changes to Retail currency exchange rates are not propagated to Oracle General Ledger. Payment terms, however, are manually set up and maintained in each system.

Oracle E-Business Suite Financials Units and Site IDs

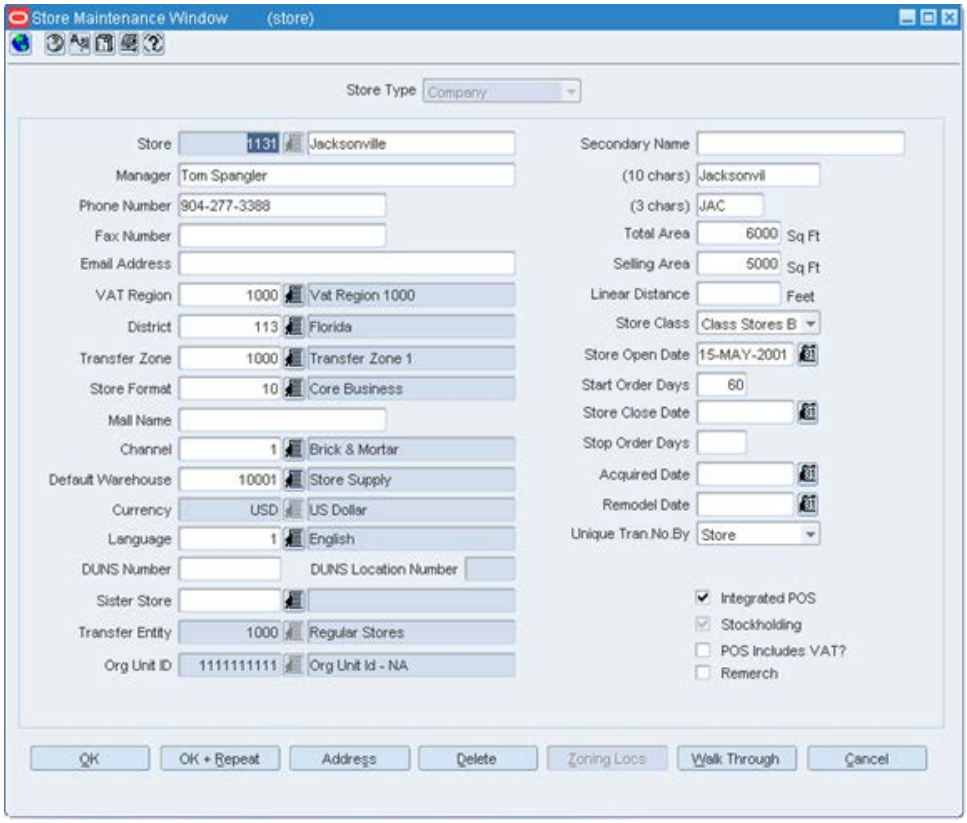
The data concepts of Org Units and Site IDs in RMS mirror the data maintained in Oracle E-Business Suite. RMS forms are used to manage and view Oracle Org Units and Site IDs. The RMS windows for Store and Warehouse maintenance allows for the association of each store and warehouse with an Org Unit. The following is an example of the Organizational Unit form:

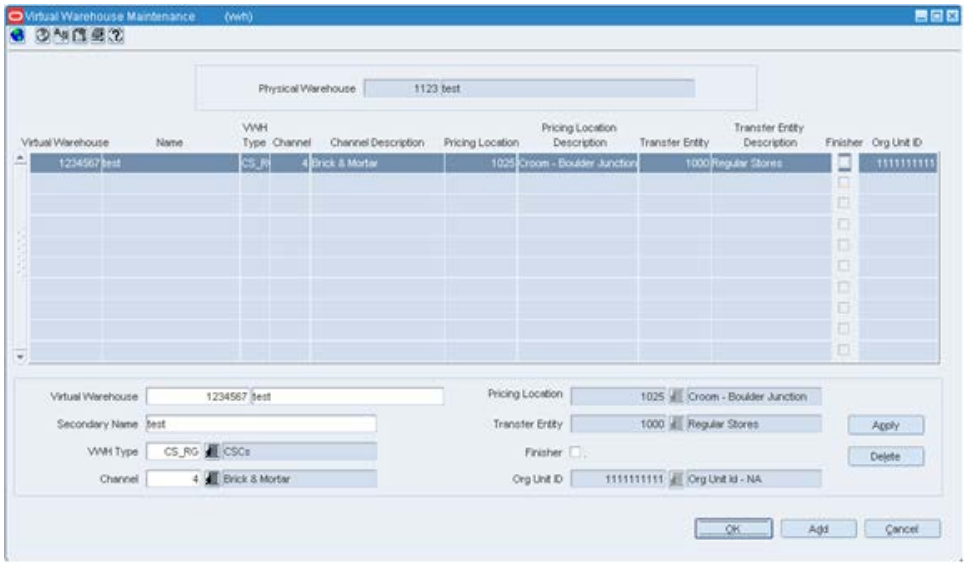


Store and Warehouse Maintenance

The organizational unit is found on the Store Maintenance and Warehouse forms, which allow the Oracle E-Business Suite operating unit to be associated with the store or warehouse. When RMS is set up for single-channel operation, the organizational unit is set at the physical warehouse level. When RMS is set up for multi-channel operation, the organizational unit is set up at the virtual warehouse level. Financial sales audit and inventory information can then be identified through interface routines and posted to the appropriate general ledger accounts. An organizational unit must be designated for each store and warehouse location in RMS.

The following are examples of the Store Maintenance and Warehouse Maintenance forms:





RMS General Ledger Setup

For RMS and ReSA, manual setup is required for validating the chart of accounts. Valid charts of accounts are created and stored in general ledger cross reference tables. Once the validation is complete, transaction data can be assigned to specific account codes.

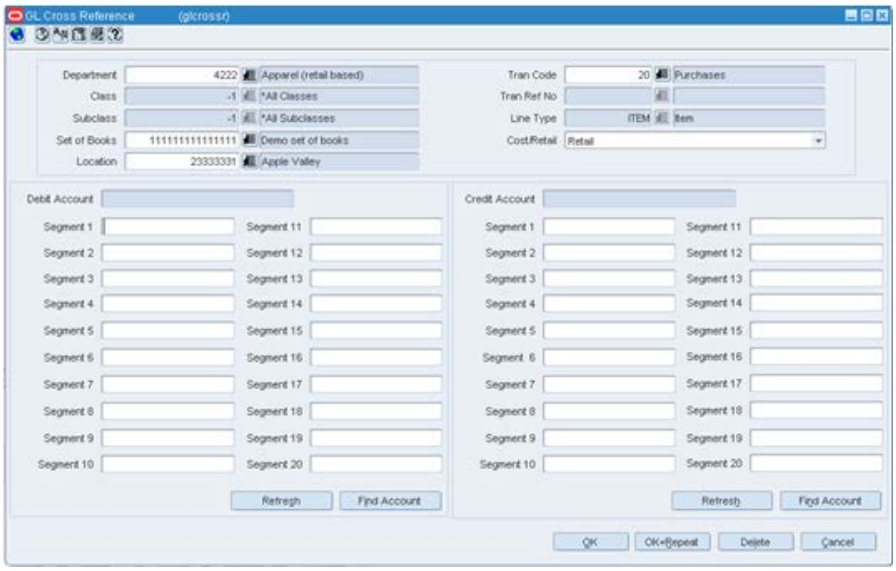
Ongoing maintenance of the chart of accounts information (such as adding, changing, or deleting chart of accounts) requires re-validation. In this regard, Oracle General Ledger is the system of record, as it is used to verify the chart of accounts used by Oracle Retail

applications. When these applications send a chart of accounts for validation, Oracle General Ledgers issues a message with a valid or invalid status, response date, and chart of accounts. The RMS table, FIF_GL_SETUP, stores the Oracle E-Business Suite Set of Books IDs to post financial information. This table must be set up manually after Set of Books IDs are determined. Where system indicator Multiple Set of Books ID is set to N, FIF_GL_SETUP must hold a single Set of Books (SOB) record.

The Set of Books IDs is associated with the chart of accounts when setting up general ledger cross reference.

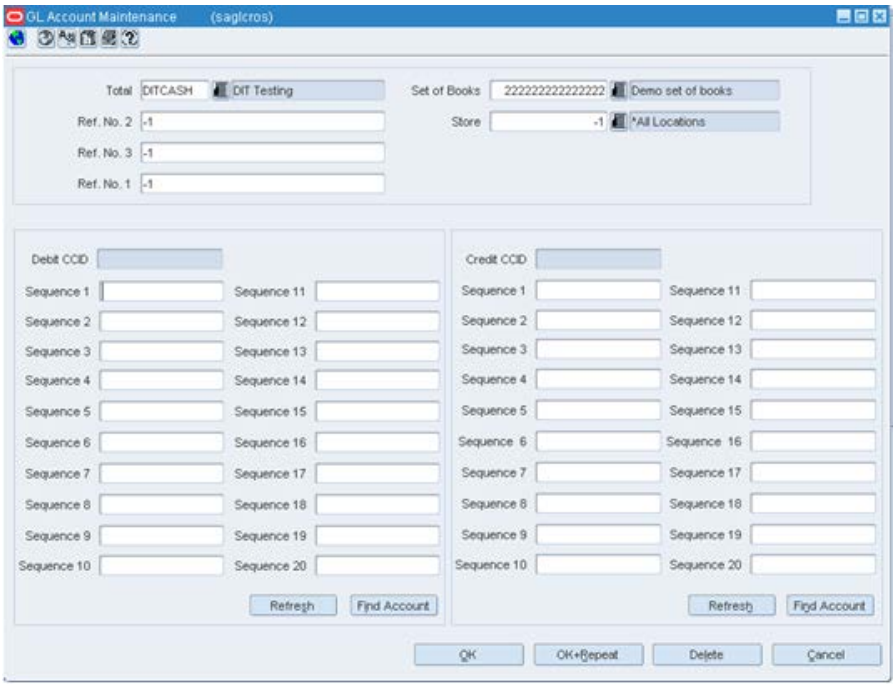
RMS General Ledger Cross Reference

Navigate: RMS main menu > Finance> GL Cross Reference. The General Ledger Search window opens. Map Chart of Accounts to department, class, subclass, set of books, location, and transaction codes using the GL cross reference form in RMS.



ReSA General Ledger Cross Reference

Navigate: ReSA main menu > Action > Sales Audit > Control > Setup > GL Account Maintenance. The General Ledger Search Form window opens. Where SYSTEM_OPTIONS.FINANCIAL_AP is A, the form requires the entry of valid segment combinations.



ReIM Data Setup and Configuration

This section describes setup considerations for ReIM data.

System Options

As part of the RMS system options setup script, set the following options as indicated:

- FINANCIAL_IND =Y
- FINANCIAL_AP =A

As part of the ReIM system options setup script, DEFAULT_PAY_NOW_TERMS should be updated with the default term ID.

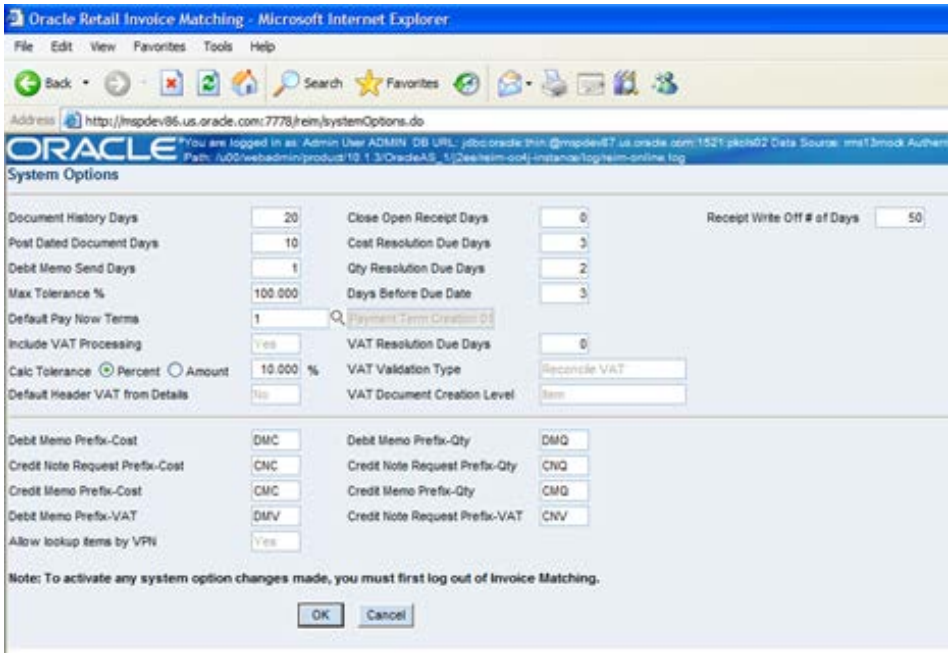
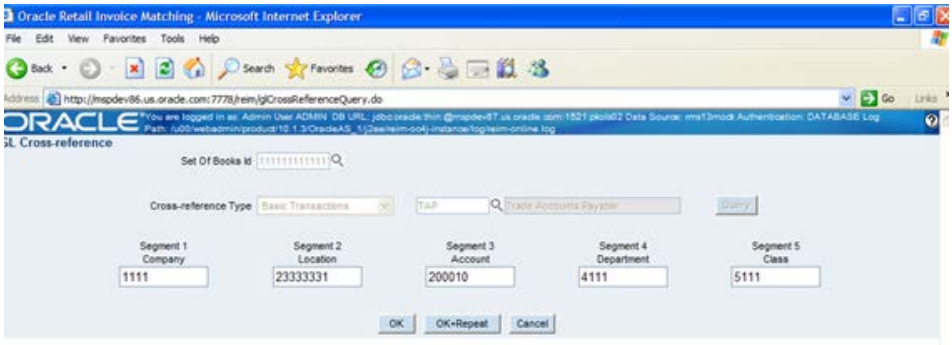


Chart of Accounts Setup

The chart of accounts is set up manually in Oracle Retail applications and in Oracle General Ledger. All account combinations are set up in each set of books. The following is an example of the GL Cross Reference screen:



Note: The Chart of accounts is updated in Oracle Retail applications only after the account is validated through Oracle General Ledger.

Segment Mapping

The retailer determines how many segments are populated. Up to 20 account segments may be specified. The following is an example of how segments are mapped between the ReIM transaction table and Oracle General Ledger:

ReIM Segments	Oracle General Ledger Chart of Accounts
Segment 1	PRODUCT
Segment 2	ACCOUNT
Segment 3	ALTACCT
Segment 4	OPERATING_UNIT
Segment 5	FUND_CODE
Segment 6	DEPTID
Segment 7	PROGRAM_CODE
Segment 8	CLASS_FLD
Segment 9	BUDGET_REF
Segment 10	BUSINESS_UNIT_PC
Segment 11	PROJECT_ID
Segment 12	ACTIVITY_ID
Segment 13	RESOURCE_TYPE
Segment 14	RESOURCE_CATEGORY
Segment 15	RESOURCE_SUB_CAT
Segment 16	CHARTFIELD1
Segment 17	CHARTFIELD2
Segment 18	CHARTFIELD3
Segment 19	AFFILIATE
Segment 20	AFFILIATE_INTRA1

If any one of the values in the 20 segments does not match the Oracle General Ledger, the account combination is considered invalid. The following error message is issued to the user: "Account combination is invalid in the financial system."

Segments 1 and 2 may be set up as dynamic at the Location level, or Segments 4 and 5 can be dynamic at the Department and Class level respectively. Segments defined as dynamic are allowed to be null for certain types of Basic Transaction or Reason Code cross-reference types. When a segment is null, the segment is assigned dynamically when transactions are posted. (Non-dynamic segments cannot be blank.) Validation applies to the segment combination, not to individual segments.

Note: For Tran code TAP, each segment must have a value regardless of whether the segment is dynamic.

Running the Initial Load from Oracle E-Business Suite Financials

The initial load for ReIM is run by Oracle E-Business Suite and includes the following information:

- Suppliers
- Currency Rates

> **Note:** The view, mv_currency_conversion_rates should be refreshed once the initial loads of currencies from Oracle General Ledger are loaded to ReIM.

integration.properties File Setup

To accommodate integration, the integration.properties file within ReIM must be updated with the appropriate URLs for the account validation and drill forward Web services, as listed below:

```
#webservice WSDL URL for drill forward
webservice.financial.drill.forward.wsdl=@webservice.drill.forward.wsdl@

webservice.financial.drill.forward.url.targetnamespace=http://oracle.apps.aia.dril
lbackforward/
webservice.financial.drill.forward.targetsystem=http://oracle.apps.aia.drillbackfo
rward/types/
#webservice WSDL URL for account validation
webservice.financial.account.validation=@webservice.account.validation@

webservice.financial.account.validation.namespace=http://xmlns.oracle.com/ABCSImpl
/Retail/Core/ProcessGLAccountValidationRetailReqABCSImpl/V1

webservice.financial.account.validation.local.code=ProcessGLAccountValidationRetai
lReqABCSImpl
#webservice username and password for account validation
webservice.financial.account.validation.username=@webservice.account.validation.us
ername@
webservice.financial.account.validation.password=@webservice.account.validation.pa
ssword
@
```

Reports are created by Business Intelligence Publisher for the following:

The URL for each report must be updated in the table, retail_service_report_url. The following table provides sample URLs:

ReIM Transactional Maintenance

Integration to Oracle General Ledger includes a number of transactions, as described below.

Calculation of TRANS_AMOUNT

The TRANS_AMOUNT field in the im_financial_stage table stores the value of the journal entry to be posted to Oracle General Ledger. (The currency for the calculated amount is the currency assigned to the transaction.) The TRANS_AMOUNT value is calculated as follows:

Row Description	DEBIT_CREDIT_IND	TRANS_AMOUNT Value
Normal	Debit	Transaction Amount
Normal	Credit	(-1) * Transaction Amount
VAT	Debit	Transaction Amount * VAT Rate
VAT	Credit	(-1) * Transaction Amount * VAT Rate

Note: Transaction Amount is taken from the database column, IM_FINANCIALS_STAGE.AMOUNT.

Generation of Outgoing Data

A staging table accommodates the outgoing transfer of data. The reference key assigned to each document or receipt is used to find data on this table

From	To	Transactions
ReIM	Oracle Payables	Invoices Debit Memos Credit Memos Credit Notes
ReIM	Oracle General Ledger	General Ledger accounting entries resulting from the Invoice Matching process, including: Pre-paid invoices Receipt Write-offs
RMS	Oracle General Ledger	Accounting entry data (potentially very high volume)
ReSA	Oracle General Ledger	Accounting entry data (potentially very high volume)

Validation of Accounts When Posting Financial Entries

Valid chart of accounts are stored in the ReIM table, IM_VALID_ACCOUNTS, which includes the Set of Books ID (sob_id) and 20 segments. An AIA Web service validates accounts against the Oracle General Ledger. Valid accounts are posted to IM_VALID_ACCOUNTS; invalid accounts are posted to IM_POSTING_DOC_ERROR. The following steps describe the validation process:

1. The ReIM system invokes the validation Web service to validate the chart of accounts. (A URL for the AIA Web service is configured in the integration.properties file.)
2. The posting batch job checks the accounts to be posted against the IM_VALID_ACCOUNTS table.
3. If the chart of accounts are in the table, the transaction is posted to staging tables.

4. If the chart of account does not exist in the table, a collection of accounts is built. These collected accounts are validated against the Oracle General Ledger, and a status is returned.
   - If the status of the collected accounts is valid, the accounts are inserted in the IM_VALID_ACCOUNTS table, and the transactions are posted to the staging tables.
   - If the status of the accounts is NOT valid, the entire collection is flagged as errors, and transactions are posted to IM_POSTING_DOC_ERROR.

> **Note:** ReIM completes the first level of account validation and posts the transaction to staging tables. It is assumed the second level of account validation is done at the end of the extraction process (where transactions are moved from ReIM staging tables to Oracle General Ledger). If account validation fails at this point, Oracle General Ledger must change the account information before transactions are loaded to Oracle General Ledger, and the chart of accounts must be re-validated in ReIM.

Maintenance of Valid Accounts

As account information is changed in the Oracle General Ledger, Retail must re-validate the locally stored chart of accounts. Oracle General Ledger will not propagate chart of account changes to Retail. The AccountPurge Batch can clear all valid accounts in the IM_VALID_ACCOUNTS table or only those that are considered updates in Oracle E-Business Suite.

Usage

```
AccountPurge userid/password PURGE [ALL | <Accounts>]
```

Where

The first argument is a combination of user id and password.

The second argument is the word PURGE.

The third argument is either ALL or specific accounts to be deleted from the local table.

11

PeopleSoft Enterprise Financials Integration

This chapter describes the integration between Oracle Retail systems and PeopleSoft Enterprise Financials, as developed and supported by Oracle Application Integration Architecture (AIA).

When the option to integrate is chosen, selected information is shared among the systems. Integration and validation services are in place to ensure the shared data matches.

The primary benefit of this integration is that clients can "drill forward" or "drill back" between the systems to research the outcome--or origin--of financial transactions. Drilling functionality is facilitated by an AIA layer, which maps, retrieves and routes information called by each system.

> **Note:** This chapter addresses the points within Oracle Retail systems that are essential to integration. For more information about the entire integration process, including mapping to PeopleSoft Enterprise Financials data and settings, see the AIA document, *Oracle Retail Merchandising Integration Pack for PeopleSoft Enterprise Financials 2.5 - Implementation Guide.* For more information about Web services, see the following chapters in th*e Oracle Retail Merchandising System Operations Guide, Volume 2*: "Service Provider Implementations API Designs" and "Web Services."

Participating Applications

The following applications are included in the integration covered by this chapter:

- Oracle Retail Merchandising System (RMS)
- Oracle Retail Sales Audit (ReSA)
- Oracle Retail Invoice Matching (ReIM)
- Oracle Retail Integration Bus

Assumptions and Dependencies

- The option to integrate should be selected during initial setup of the RMS system.
- ReIM accesses RMS to determine if integration is active. Initial setup of RMS must occur prior to the integration of ReIM.
- When a PeopleSoft user performs a drill back, data is presented through an Oracle Business Intelligence Publishing report.
- When drilling forward to PeopleSoft Enterprise Financials, users have view-only access.
- The URLs for the AIA Web services that are anecessary for this integration must be maintained in the RMS_RETAIL_SERVICE_REPORT_URL table and in the ReIM integration.properties file.
- Real time account validation is done only when the financial integration with PeopleSoft Enterprise Financials is ON.

- Partners must be set up as suppliers in PeopleSoft. Then the partner must be manually set up in RMS using the RMS Supplier ID that was generated when the PeopleSoft supplier was interfaced to Oracle Retail. Partner functionality within RMS and ReIM can then proceed normally. The RMS supplier generated as part of this process is not used.

Data Constraints

- The Location ID field is restricted to eight characters, to accommodate PeopleSoft Operating Unit, which has a maximum of eight characters.
- The Ext_Doc_ID field is restricted to 30 characters, because the corresponding PeopleSoft field has only 30 characters. Characters beyond 30 are truncated.
- RMS allows for four decimals, and PeopleSoft allows only three. Truncation may occur when data is passed to PeopleSoft Enterprise Financials.
- ReIM values in the IM_CURRENCY_LOCALE are restricted to three decimals, because the corresponding PeopleSoft Enterprise Financials field can accept no more than three decimal positions.

Data Setup

Integration of Oracle Retail and PeopleSoft Enterprise Financials relies on synchronization of essential data, such as rates and terms. Through careful discussions, users of both systems determine the common codes and descriptions that will best serve their business needs.

Once agreement is reached, this information is set up and maintained. Depending on volume, some shared information is set up in either Oracle Retail or PeopleSoft Enterprise Financials-and electronically transferred to the other system. Otherwise, shared information is set up manually within each system, and users of both systems must ensure that codes and descriptions match.

RMS Data Setup and Configuration

RMS System Options

As part of the RMS system options setup, set the following options as indicated:

- FINANCIAL_IND = Y

  This system_option indicates that the Oracle Retail system is integrated with a financial system:
- FINANCIAL_AP = A

  A value of A indicates that the financial system to which RMS is interfaced is Oracle Peoplesoft Enterprise Financials through Oracle Application Integration Architecture (AIA).
- GL_ROLL_UP can be D/S/C
- MULTIPLE_SET_OF_BOOKS_IND = Y
- SUPPLIER_SITE_IND = Y
- ORG_UNIT_IND = Y

Organization Units

Use the Organizational Unit window (RMS Start Menu > Control > Setup > Org Unit >Edit) to define organizational units in RMS that match those being set up in Peoplesoft Enterprise Financials. When an organizational unit is entered in RMS, the valid organizational units are those that are associated with the set of books (SOB) that is being used for the general ledger interface.

Currency Exchange Rates

Currency exchange rate is used to translate the monetary value of one currency in terms of another. Depending on business needs, a Currency Exchange Rate Type of Operational or Consolidation is selected for use in all transactions.

This value is set up manually in RMS and mapped to PeopleSoft Enterprise Financials through the Currency Exchange Type mapping Window. Currency Exchange Rate data is owned by PeopleSoft Enterprise Financials, and updates are sent to Oracle Retail applications.

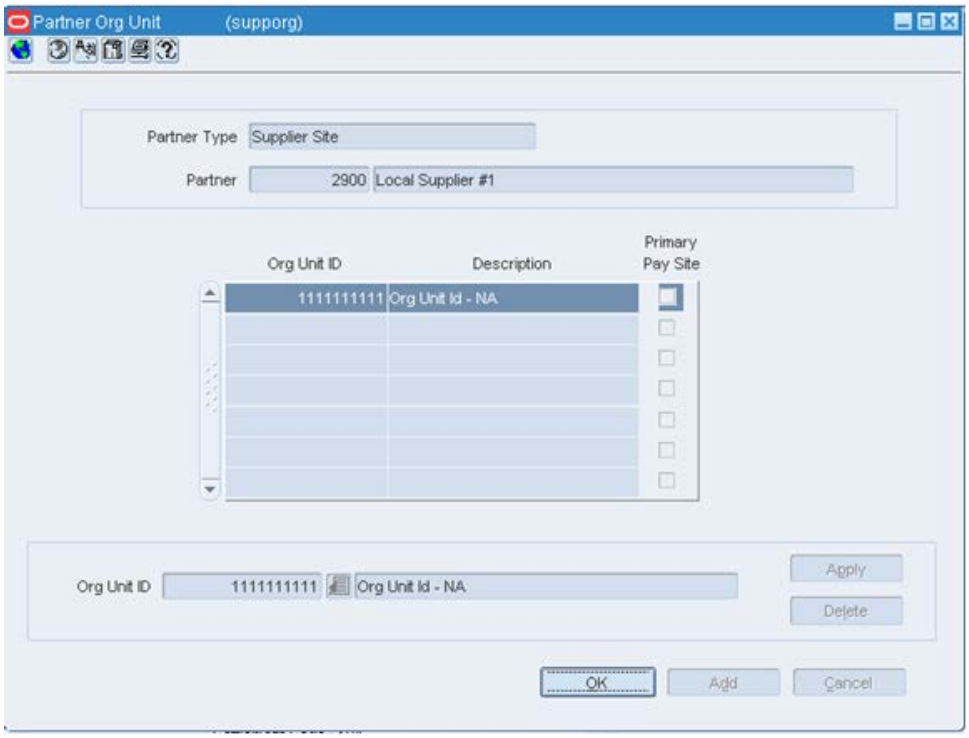
Determine the Exchange Type being sent by Oracle PeopleSoft Financials (for example, Consolidation or Operational) that you want RMS to use. Then update the FIF_CURRENCY_XREF for mapping the external exchange type being sent by Oracle Peoplesoft Financials with RMS Exchange Type.

For example, for Consolidation and Operational exchange types, the FIF_CURRENCY_XREF table holds the following entries:

FIF_EXCHANGE_TYPE	RMS_EXCHANGE_TYPE
C	C
O	O

Supplier Address Types

Within RMS, supplier information (such as Order From and Remit To addresses) is used in generating purchase orders. PeopleSoft uses supplier information for payment generation. It is important, then, that this information is synchronized.



Suppliers are created in Peoplesoft Enterprise Financials and exported to RMS. When FINANCIAL_AP is set to A, suppliers cannot be created using the RMS forms. However, after the supplier exists in RMS, all data values for the supplier (except supplier name and status) continue to be updated using the RMS forms. The association of supplier sites to organization units is accessed only in view mode through RMS forms. One supplier site per supplier organization unit combination can be marked as primary payment site.

Where SYSTEM_OPTIONS.FINANCIAL_AP is A, disable auto generate supplier/partner numbers and associated check boxes.

> **Note:** Supplier information is created, updated and inactivated only in the PeopleSoft Enterprise Financials accounting system. This information is transferred from PeopleSoft Enterprise Financials to Oracle Retail, where additional retail-specific attributes may be maintained.

Country Codes

When country codes are defined and seeded in RMS, ensure that country codes are mapped to PeopleSoft country codes through AIA DVM mapping. The following is an example of AIA DVM Mapping for COUNTRY_CODE:

COUNTRY_CODE		
PSFT_01	**COMMON**	**RETL_01**
USA	700	US
CAN	701	CA

Note: For more mapping examples, see AIA DVM Mapping Examples later in this chapter.

Financial Calendar

The financial calendar within Oracle Retail systems is manually set up and maintained separately from the PeopleSoft financial calendar.

Freight Terms

A freight term is an agreement between the retailer and a supplier regarding transportation charges for goods delivered by the supplier. Freight terms are used by RMS as purchase orders are generated.

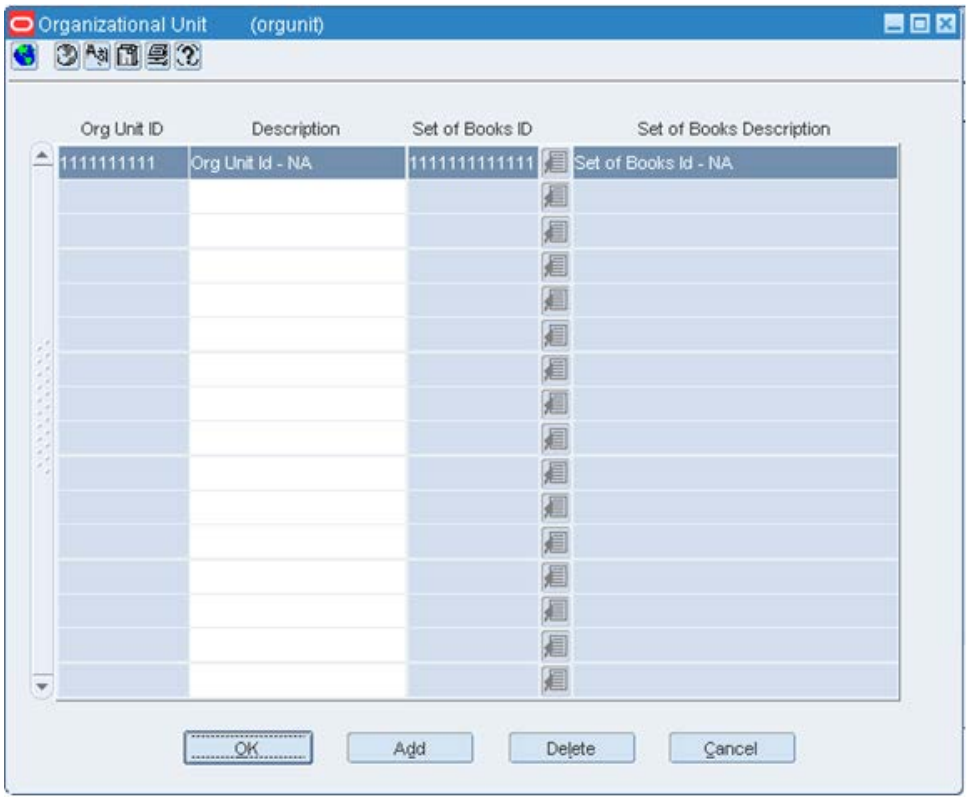
Within the RMS system, freight terms are set up and maintained manually. They also are maintained in PeopleSoft Enterprise Financials.

Payment Terms and Currency Exchange Rates

This data is created and updated in Oracle Peoplesoft Financials and exported to RMS. It is not created or updated in RMS.

PeopleSoft Enterprise Financials Org Units and Site IDs

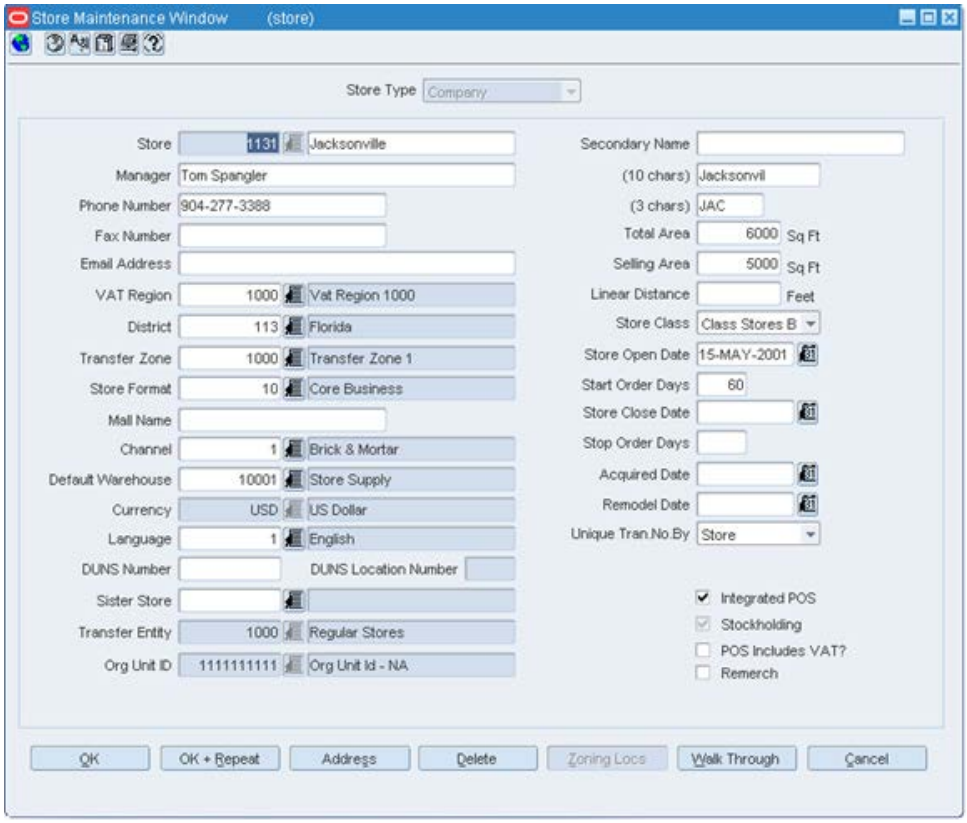
The data concepts of Org Units and Site IDs in RMS mirror the data maintained in Oracle PeopleSoft Enterprise Financials. RMS forms are used to manage and view Oracle Org Units and Site IDs. The RMS windows for Store and Warehouse maintenance allows for the association of each store and warehouse with an Org Unit. The following is an example of the Organizational Unit form:

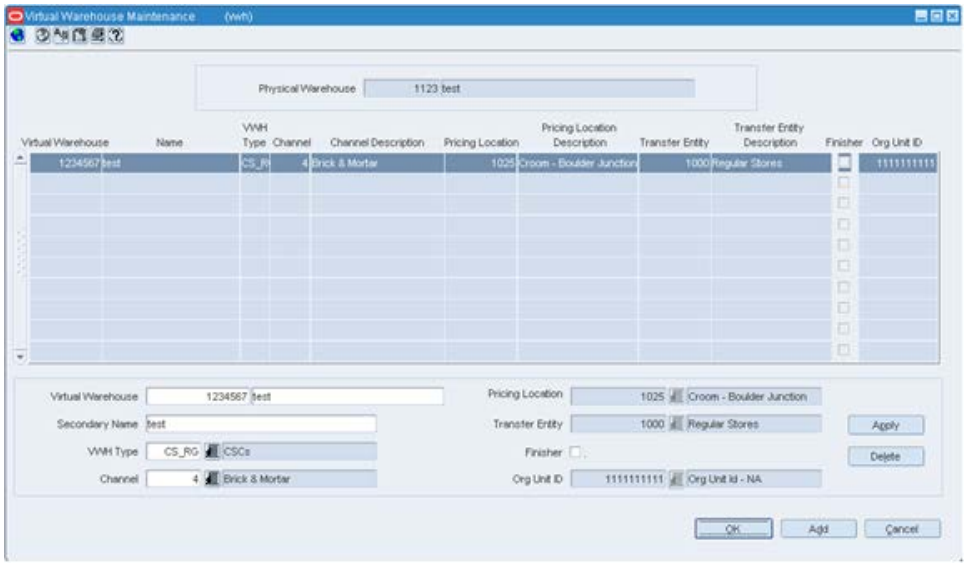


Store and Warehouse Maintenance

The organizational unit is found on the Store Maintenance and Warehouse forms, which allow the PeopleSoft Enterprise Financials operating unit to be associated with the store or warehouse. When RMS is set up for single-channel operation, the organizational unit is set at the physical warehouse level. When RMS is set up for multi-channel operation, the organizational unit is set up at the virtual warehouse level. Financial sales audit and inventory information can then be identified through interface routines and posted to the appropriate general ledger accounts. An organizational unit must be designated for each store and warehouse location in RMS. The following are examples of the Store Maintenance and Warehouse Maintenance forms:

> **Note:** When working with PeopleSoft Enterprise Financials, be sure to create locations names (such as stores, WH, and virtual WH) with eight or fewer characters. Locations in Oracle Retail applications are mapped to Operating Unit in PeopleSoft Enterprise Financials, where operating unit cannot exceed eight characters.





RMS General Ledger Setup

Within RMS and ReSA, manual setup is required for the initial load of chart of accounts information, including valid segment combinations of Chart of Accounts segment/ChartField values for each business unit. Valid segment combinations are created and stored in general ledger cross reference tables. Once setup is complete, transaction data can be assigned to specific account codes.

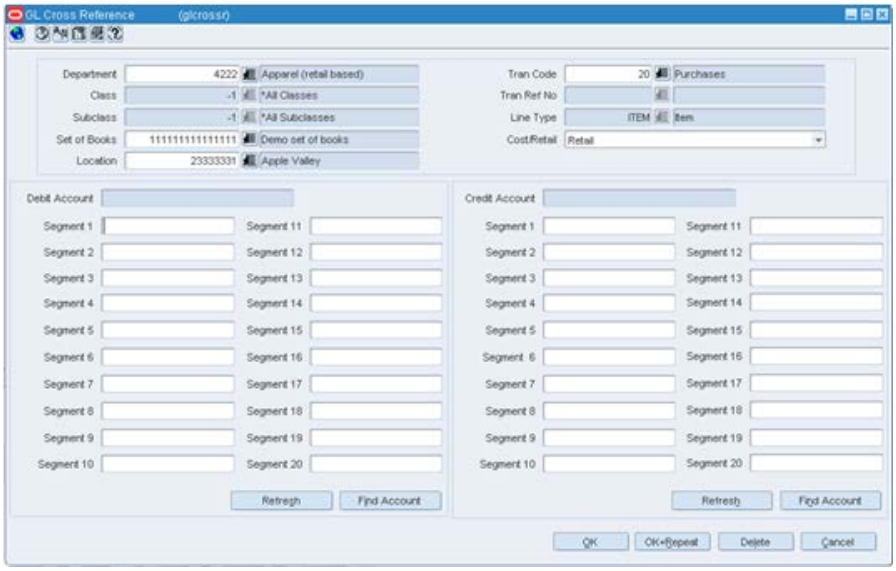
Ongoing maintenance of Chart of Accounts information (such as adding, changing or deleting segment/ChartField values) also is completed manually. Any segment combination that is valid in Oracle Retail applications also must be valid in PeopleSoft Enterprise Financials. In this regard, PeopleSoft Enterprise Financials is the system of record, in that it verifies segment combinations created or updated within Oracle Retail. PeopleSoft Enterprise Financials issues a message when an Oracle Retail segment combination is invalid, and the retail user must correct the appropriate cross reference table.

The RMS table FIF_GL_SETUP holds the PeopleSoft Enterprise Financials Set of Books IDs to post financials. This table requires manual setup after the Set of Books IDs are determined. Where system indicator Multiple Set of Books ID is set to N, FIF_GL_SETUP must hold a single Set of Books (SOB)record.

The Set of Books IDs is associated with the chart of accounts when setting up general ledger cross reference.

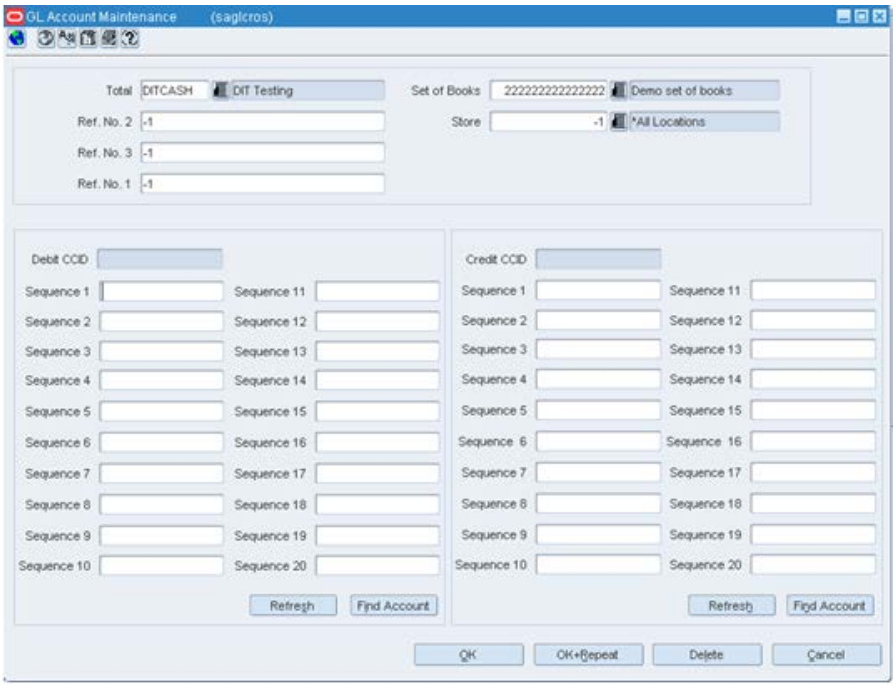
RMS General Ledger Cross Reference

Navigate: RMS main menu > Finance> GL Cross Reference. The General Ledger Search window opens. Map Chart of Accounts to department, class, subclass, set of books, location, and transaction codes using the GL cross reference form in RMS.



ReSA General Ledger Cross Reference

Navigate: ReSA main menu > Action > Sales Audit > Control > Setup > GL Account Maintenance. The General Ledger Search Form window opens. Where SYSTEM_OPTIONS.FINANCIAL_AP is A, the form requires the entry of valid segment combinations.



Configuring Drill Back and Forward Web Services

Retail web services table, RETAIL_SERVICE_REPORT_URL, must be updated with appropriate URLs to integrate with PeopleSoft Enterprise Financials.

- The records in the table for Services (indicated by RS_TYPE=S) for Account Validation (RAV) and Drill Forward (RDF), must be updated with the URL information from AIA where the services are hosted.

  Note: If Web services are secure, then the SYS_ACCOUNT column must be populated with authentication information in the form of u*sername/password*.

- The records in the table for Reports (indicated by RS_TYPE=R) for both RMS and ReIM reports, must be updated with the URL information from the BIP Server where the reports are hosted.

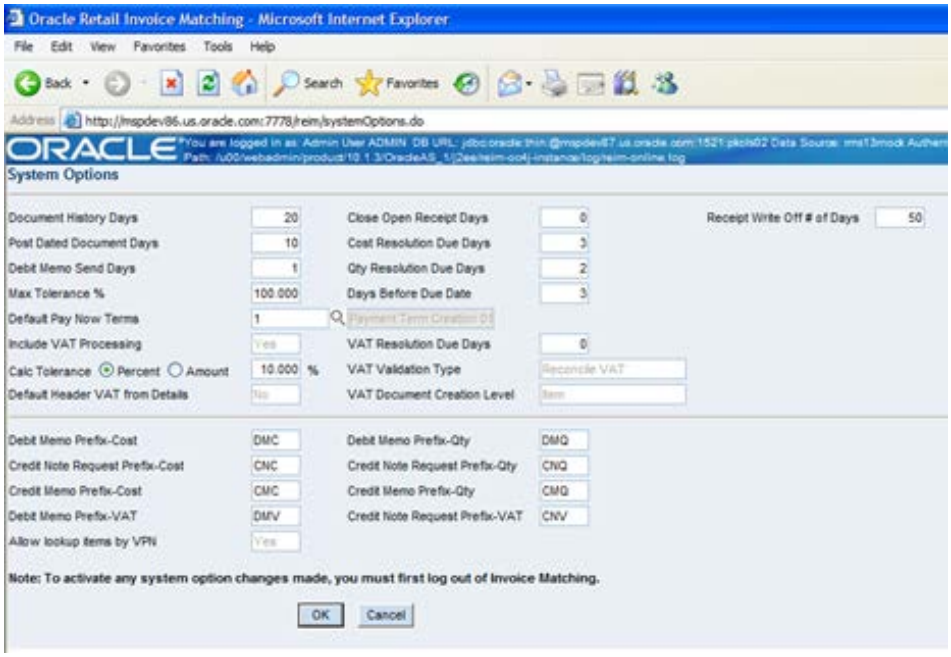
ReIM Data Setup and Configuration

System Options

As part of the RMS system options setup script, set the following options as indicated:

- FINANCIAL_IND = Y
- FINANCIAL_AP = A

As part of the ReIM system options setup script, DEFAULT_PAY_NOW_TERMS should be updated with the default term ID.



IM_CURRENCY_LOCALE

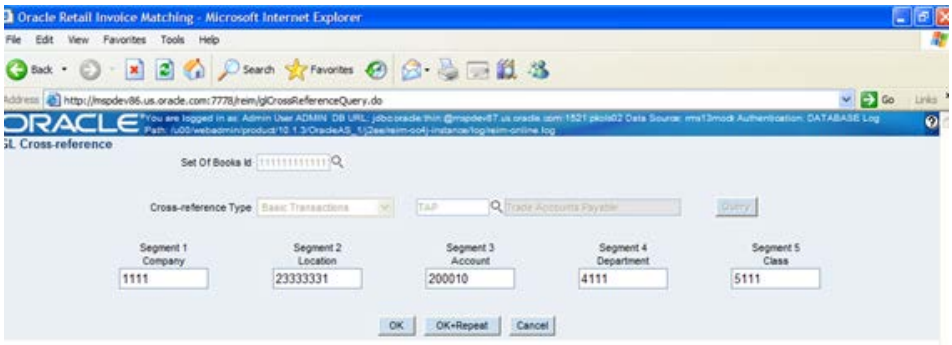
Because PeopleSoft Enterprise Financials uses only three decimals, the transactions generated by the Oracle Retail ReIM application must not include more than three decimals.

Update im_currency_locale set currency_cost_dec=3.

Prerequisite: The currency_rates table in RMS should be loaded initially by PeopleSoft Enterprise Financials.

Chart of Accounts Setup

The chart of accounts is set up manually in Oracle Retail applications and in PeopleSoft Enterprise Financials. All account combinations are set up in each Set of Books. The following is an example of the GL Cross Reference screen:



Note: Chart of Accounts is updated in Oracle Retail applications only after the account is validated through PeopleSoft Enterprise Financials.

Segment Mapping

The retailer determines how many segments are populated. Up to 20 account segments may be specified. The following is an example of how segments are mapped between the ReIM transaction table and PeopleSoft Enterprise Financials:

ReIM Segments	PeopleSoft Enterprise Financials Fields
Segment 1	PRODUCT
Segment 2	ACCOUNT
Segment 3	ALTACCT
Segment 4	OPERATING_UNIT
Segment 5	FUND_CODE
Segment 6	DEPTID
Segment 7	PROGRAM_CODE
Segment 8	CLASS_FLD
Segment 9	BUDGET_REF
Segment 10	BUSINESS_UNIT_PC
Segment 11	PROJECT_ID
Segment 12	ACTIVITY_ID
Segment 13	RESOURCE_TYPE
Segment 14	RESOURCE_CATEGORY
Segment 15	RESOURCE_SUB_CAT
Segment 16	CHARTFIELD1
Segment 17	CHARTFIELD2
Segment 18	CHARTFIELD3
Segment 19	AFFILIATE
Segment 20	AFFILIATE_INTRA1

If any one of the values in the 20 segments does not match the corresponding PeopleSoft field value, the account combination is considered invalid. The following error message is issued to the user: "Account combination is invalid in the financial system."

Segments 1 and 2 may be set up as dynamic at the Location level, or Segments 4 and 5 can be dynamic at the Department and Class level respectively. Segments defined as dynamic are allowed to be null for certain types of Basic Transaction or Reason Code cross-reference types. When a segment is null, the segment is assigned dynamically when transactions are posted. (Non-dynamic segments cannot be blank.) Validation applies to the segment combination, not to individual segments.

> **Note:** For Tran code TAP, all segments must have a value regardless of whether the segment is dynamic.

Running the Initial Load from PeopleSoft Enterprise Financials

The initial load for ReIM is run by PeopleSoft Enterprise Financials and includes the following information:

- Suppliers
- Payment Terms
- Currency Rates

> **Note:** The view, mv_currency_conversion_rates should be refreshed once the initial loads of currencies from PeopleSoft Enterprise Financials are loaded to ReIM.

integration.properties File Setup

To accommodate integration, the integration.properties file within ReIM must be updated with the appropriate URLs for the account validation and drill forward Web services, as listed below:

```
#webservice provider URL for drill forward
webservice.financial.drill.forward=@webservice.drill.forward@

#webservice provider URL for account validation
webservice.financial.account.validation=@webservice.account.validation@

#webservice username and password for account validation
webservice.financial.account.validation.username=@webservice.account.validation.us
ername@
webservice.financial.account.validation.password=@webservice.account.validation.pa
ssword
@
```

Reporting

Reports are created by Business Intelligence Publisher for the following:

- Merchandise Invoice
- Non-Merchandise Invoice
- Credit Note
- Credit Memo
- Debit Memo
- Receipt Write-Off

The URL for each report must be updated in the table, retail_service_report_url. The following table provides sample URLs:

Document Type	Report Name	Sample Report URL
MRCHI	Merchandise invoice document Report	`http://mspdv126.us.oracle.com:7777/xmlpserver_nonsso/Guest/REIM13/Finance/invreport/invreport.xdo`
NMRCHI	Non-Merchandise invoice document Report	`http://mspdv126.us.oracle.com:7777/xmlpserver_nonsso/Guest/REIM13/Finance/invreport/invreport.xdo`

Document Type	Report Name	Sample Report URL
CRDNT	Credit Note document Report	http://mspdv126.us.oracle.com:7777/xmlpserver_nonsso/Guest/REIM13/Finance/invreport/crnreport.xdo
CRDMEC	Credit Memo cost document Report	http://mspdv126.us.oracle.com:7777/xmlpserver_nonsso/Guest/REIM13/Finance/invreport/memoreport.xdo
CRDMEQ	Credit Memo quantity document Report	http://mspdv126.us.oracle.com:7777/xmlpserver_nonsso/Guest/REIM13/Finance/invreport/memoreport.xdo
DEBMEC	Debit Memo cost document Report	http://mspdv126.us.oracle.com:7777/xmlpserver_nonsso/Guest/REIM13/Finance/invreport/imemoreport.xdo
DEBMEQ	Debit Memo quantity document Report	http://mspdv126.us.oracle.com:7777/xmlpserver_nonsso/Guest/REIM13/Finance/invreport/memoreport.xdo
DEBMEV	Debit Memo Tax document Report	http://mspdv126.us.oracle.com:7777/xmlpserver_nonsso/Guest/REIM13/Finance/invreport/memoreport.xdo
RWO	Receipt Write Off document Report	http://mspdv126.us.oracle.com:7777/xmlpserver_nonsso/Guest/REIM13/Finance/invreport/rworeport.xdo

ReIM Transactional Maintenance

Integration to PeopleSoft Enterprise Financials includes a number of transactions, as described below.

Calculation of TRANS_AMOUNT

The TRANS_AMOUNT field in the im_financial_stage table stores the value of the journal entry to be posted to PeopleSoft Enterprise Financials. (The currency for the calculated amount is the currency assigned to the transaction.) The TRANS_AMOUNT value is calculated as follows:

Row Description	DEBIT_CREDIT_IND	TRANS_AMOUNT Value
Normal	Debit	Transaction Amount
Normal	Credit	(-1) * Transaction Amount
VAT	Debit	Transaction Amount * TaxRate
VAT	Credit	(-1) * Transaction Amount * Tax Rate

Note: Transaction Amount is taken from the database column, IM_FINANCIALS_STAGE.AMOUNT.

Generation of Outgoing Data

A staging table accommodates the outgoing transfer of data. The reference key assigned to each document or receipt is used to find data on this table.

Outgoing Data

From	To	Transactions
ReIM	PeopleSoft Accounts Payable	Invoices Debit Memos Credit Memos Credit Notes
ReIM	PeopleSoft General Ledger	General Ledger accounting entries resulting from the Invoice Matching process, including: Pre-paid invoices Receipt Write-offs
RMS	PeopleSoft General Ledger	Accounting entry data (potentially very high volume)
ReSA	PeopleSoft General Ledger	Accounting entry data (potentially very high volume)

Validation of Accounts When Posting Financial Entries

Valid accounts are stored in the ReIM table, IM_VALID_ACCOUNTS, which includes the Set of Books ID (sob_id) and 20 segments. An AIA Web service validates accounts against the PeopleSoft Enterprise Financials system. Valid accounts are posted to IM_VALID_ACCOUNTS; invalid accounts are posted to IM_POSTING_DOC_ERROR. The following steps describe the validation process:

1. The ReIM system invokes the account validation Web service to validate the account. (A URL for the AIA Web service is configured in the integration.properties file.)
2. The posting batch job checks the accounts to be posted against the IM_VALID_ACCOUNTS table.

3. If the account entries are in the table, the transaction is posted to the G/L or AP tables.
4. If the account does not exist in the table, a collection of accounts is built. These collected accounts are validated against the PeopleSoft Enterprise Financials system, and a status is returned.
   - If the status of the collected accounts is valid, the accounts are inserted in the IM_VALID_ACCOUNTS table, and the transactions are posted to the staging tables.
   - If the status of the accounts is NOT valid, the entire collection is flagged as errors, and transactions are postedto IM_POSTING_DOC_ERROR.

> **Note:** ReIM completes the first level of account validation and posts the transaction to staging tables. It is assumed the second level of account validation is done at the end of the extraction process (where transactions are moved from ReIM staging tables to PeopleSoft). If account validation fails at this point, Oracle Data Integrator (ODI) or PeopleSoft must change the account information before transactions are loaded to PeopleSoft, and the account change must be communicated to ReIM .

Maintenance of Valid Accounts

As account information is changed in the PeopleSoft system, the same changes are communicated to, and manually completed in, the ReIM system. After ReIM is updated accordingly, the AccountPurge Batch is run to clear the valid accounts maintained locally in ReIM.

The AccountPurge Batch can clear all valid accounts in the IM_VALID_ACCOUNTS table or only those that are considered updates in PeopleSoft.

Usage:

```
AccountPurge userid/password PURGE [ALL | <Accounts>]
```

Where

The first argument is a combination of user id and password.

The second argument is the word PURGE.

The third argument is either ALL or specific accounts to be deleted from the local table.

Building and Posting Reference IDs

Drill back and drill forward functionality uses Reference ID to locate documents and receipts. A Reference ID is a combination of document type and document (or receipt) ID, as illustrated in the table below:

Type	Doc ID	Receipt ID	Reference ID
Merchandise Invoice	101	Null	MRCHI#101
Non-Merchandise Invoice	102	Null	NMRCHI#102
Receipt	Null	103	RECEIPT#103

For documents, the Resolution Posting Batch program builds the Reference ID using the standard, Document Type + DeLimiter + Doc_id. For receipts, the program builds the Reference ID using the standard, Document Type + DeLimiter + Receipt_id.

To enable drill down functionality, Reference IDs are loaded to staging tables. FinancialsAPStageDao and FinancialsGLStageDao are populated, as are IM_RWO_SHIPMENT_HIST and IM_RWO_SHIPSKU_HIST.

Drilling Back to RMS, ReSA and ReIM - Overview

Drilling back allows users to view the source of posted PeopleSoft transactions that originated in Oracle Retail systems (from a voucher to an invoice, for example).

When drilling back from PeopleSoft Enterprise Financials, users are not directed to an actual screen within RMS, ReIM or ReSA. Rather, a retail Web service generates and launches a URL to a BI Publisher report. The report contains the information that typically appears on the appropriate retail screen.

Depending on the information requested by the user, the AIA layer maps a common ID (for example, Transaction ID or Document ID) to the appropriate retail application. As part of the extraction process, the Reference_ID in the staging tables is considered the retail key, which is sent to AIA. The AIA layer invokes the retail exposed Web service for the BI Publisher report URL using a common function like the following:

```
GET_REPORT_URL() -
     O_error_message     IN OUT     RTK_ERRORS.RTK_TEXT%TYPE
     O_rpt_url           IN OUT     RETAIL_SERVICE_REPORT_URL.URL%TYPE
     I_ref_key           IN         KEY_MAP_GL.REFERENCE_TRACE_ID%TYPE
```

Information from the reference key determines what kind of report URL to issue. For example, if the retail key has a prefix of RMS, the RMS_REPORT_URL function is called. Similarly, if the retail key has a prefix of ReIM, the ReIM_REPORT_URL function is called to retreive the appropriate ReIM URL. If the key does not have a prefix (or does not match any key in the retail systems), an error message is launched.

Drilling Back to RMS and ReSA - Detail

The following function determines which RMS report to return to the user:

```
RMS_REPORT_URL() -
     O_error_message     IN OUT     RTK_ERRORS.RTK_TEXT%TYPE
     O_rpt_url           IN OUT     RETAIL_SERVICE_REPORT_URL.URL%TYPE
     I_ref_key           IN         KEY_MAP_GL.REFERENCE_TRACE_ID%TYPE
```

The appropriate report URL is found and issued as follows:

1. The ref_trace_type is found on KEY_MAP_GL by matching I_ref_key with the KEY_MAP_GL.REFERENCE_TRACE_ID column.
2. When ref type is determined, the re_trace_type is used to find the appropriate report URL on the RETAIL_SERVICE_REPORT_URL table.
3. The value of I_ref_key is appended to the end of the URL retrieved from the table.
4. The URL is sent back to the calling function.
5. If I_ref_key does not exist on KEY_MAP_GL, an error message is sent back to the calling function.

Drilling Back From ReIM - Detail

The following drill back options are available for viewing information within the ReIM system:

- Using Document ID, users can drill back to ReIM to view information related to a voucher or payment. The report includes information from im_doc_head and

im_invoice_detail, the same data shown on the Document Maintenance Header screen within ReIM.

- Using the Receipt ID, users can drill back to view information from the Receipt Write-off History screen. Receipt write-offs occur either when an open receipt is closed in ReIM or if a receipt is purged in RMS before it is fully matched. Details come from the IM_RWO_SHIPMENT_HIST and IM_RWO_SHIPSKU_HIST tables.

The function below determines which of the two ReIM reports to return to the user:

```
RMS_REPORT_URL() -
     O_error_message     IN OUT     RTK_ERRORS.RTK_TEXT%TYPE
     O_rpt_url           IN OUT     RETAIL_SERVICE_REPORT_URL.URL%TYPE
     I_ref_key           IN         KEY_MAP_GL.REFERENCE_TRACE_ID%TYPE
```

The I_ref_key contains the reference ID, which ultimately determines the type of report required. The appropriate BI Publisher report URL is found on the RETAIL_SERVICE_REPORT_URL table.

In general, if the reference ID has a prefix of RECEIPT, the report type (RS_CODE) is RCPT. Otherwise, the report type is DOC. For example:

Reference ID	Report Type (RS_CODE)
MRCHI#101	DOC
NMRCHI#102	DOC
RECEIPT#103	RCPT

The following is an example of a BI Publisher URL that is generated upon drilling back to PeopleSoft Enterprise Financials for information on an invoice in ReIM, using Document ID as the search parameter:

```
http://mspdev6970vip:7777/BIPublisher/Guest/ReIM/13.0.3/doc/tsf_det.xdo
?doc_id=101
```

Where

- http://mspdev6970vip:7777/BIPublisher = the BI Publisher application server address and port
- Guest/ReIM/13.0.3 = the directory/folder location
- doc/tsf_det.xdo ? = report name (Document Report)
- doc_id=101 = the parameter name and value (Document ID 101)

The following is an example of an Oracle Business Intelligence Publisher URL that is generated upon drilling back to PeopleSoft Enterprise Financials for information on an invoice in ReIM, using Receipt ID as the search parameter:

```
http://mspdev6970vip:7777/BIPublisher/Guest/ReIM/13.0.3/doc/tsf_det.xdo
?receipt_id=101
```

Where

- http://mspdev6970vip:7777/BIPublisher = the BI Publisher application server address and port
- Guest/ReIM/13.0.3 = the directory/folder location
- doc/tsf_det.xdo ? = report name (Receipt Report)
- receipt_id=101 = the parameter name and value (Receipt ID 101)

Drilling Forward

Drilling forward allows users to see detailed information about retail transactions that have been posted to PeopleSoft Enterprise Financials. When drilling forward, users are directed to selected "view-only" screens.

Drilling Forward From RMS/ReSA to PeopleSoft Enterprise Financials

The following forms may be used to drill forward from RMS/ReSA:

- RMS StartMenu->Finance->Transaction Data View (trandata.fmb)
- RMS StartMenu->Ordering->Fixed Deals->Fixed Deal Transaction Data View (fdltrandata.fmb)
- RMS StartMenu->Action->Sales Audit->Sales Audit Transaction Data View (satrandata.fmb)

Drilling Forward From ReIM to PeopleSoft Enterprise Financials

For drilling forward, the AIA Web service uses the Invoice ID and Accounting Entry parameters. The ReIM system uses these parameters together as the Reference ID.

- From the Document Maintenance screen, users can drill forward to PeopleSoft Enterprise Financials accounts payable to view voucher and payment status. The information is displayed on a read-only Payment Doc Status Inquiry screen. Drill forward access to the accounts payable system is available only for pre-paid invoices (but not for manually pre-paid invoices).

  To drill down to the payables screens, the user invokes the Web service as follows:

  ```
  Invoice ID parameter = Reference ID
  Accounting Entry parameter = null
  ```

- From the Receipt Write-off History screen in ReIM, users can drill forward to the PeopleSoft G/L system to view the status of related journal entries. Drill forward access to the G/L system is available only for pre-paid and manually pre-paid invoices.

> **Note:** For more information on drilling forward, see the *OracleRetail Merchandising Integration Pack for PeopleSoft Enterprise Financials: Financial Operations Control 2.5 - Implementation Guide*.

AIA DVM Mapping Examples

The following tables illustrate possible AIA DVM Mapping values:

> **Note:** Cross references for the following examples are: Set of Books to General Ledger Business Unit, and Org Unit to Accounts Payable Business Unit.

BUSINESS_UNIT		
PSFT_01	**COMMON**	**RETL_01**
US001	100	111111111111111
CAN01	101	222222222222222

BUSINESS_UNIT_AP		
PSFT_01	**COMMON**	**RETL_01**
US001	200	333333333333333
US002	201	444444444444444
CAN01	202	555555555555555

LANGUAGE_CODE		
PSFT_01	**COMMON**	**RETL_01**
ENG	500	1

STATE		
PSFT_01	**COMMON**	**RETL_01**
CA	900	CA
NY	901	NY
MD	902	MD
MI	903	MI
NF	904	NF

CURRENCYEXCHANGE_CONVERSIONTYPECODE		
PSFT_01	**COMMON**	**RETL_01**
CRRNT	600	C

CURRENCY_CODE		
PSFT_01	**COMMON**	**RETL_01**
USD	400	USD
CAD	401	CAD
AUD	402	AUD
ESP	403	ESP
EUR	404	EUR
FRF	405	FRF
GBP	406	GBP
INR	407	INR
JPY	408	JPY

SUPPLIERPARTY_STATUSCODE		
PSFT_01	**COMMON**	**RETL_01**
E	800	I
A	801	A
I	802	I
X	803	I

SUPPLIERPARTY_ADDRESSTYPE		
PSFT_01	**COMMON**	**RETL_01**
REMT	1000	06
ORDR	1001	04

CHARTOFACCOUNTS_ACCOUNTSTATUS		
COMMON	**PSFT_01**	**RETL_01**
true	true	valid
false	false	invalid

12

Setting up Oracle Business Intelligence Publisher

RMS uses Oracle Business Intelligence (BI) Publisher for publishing reports.

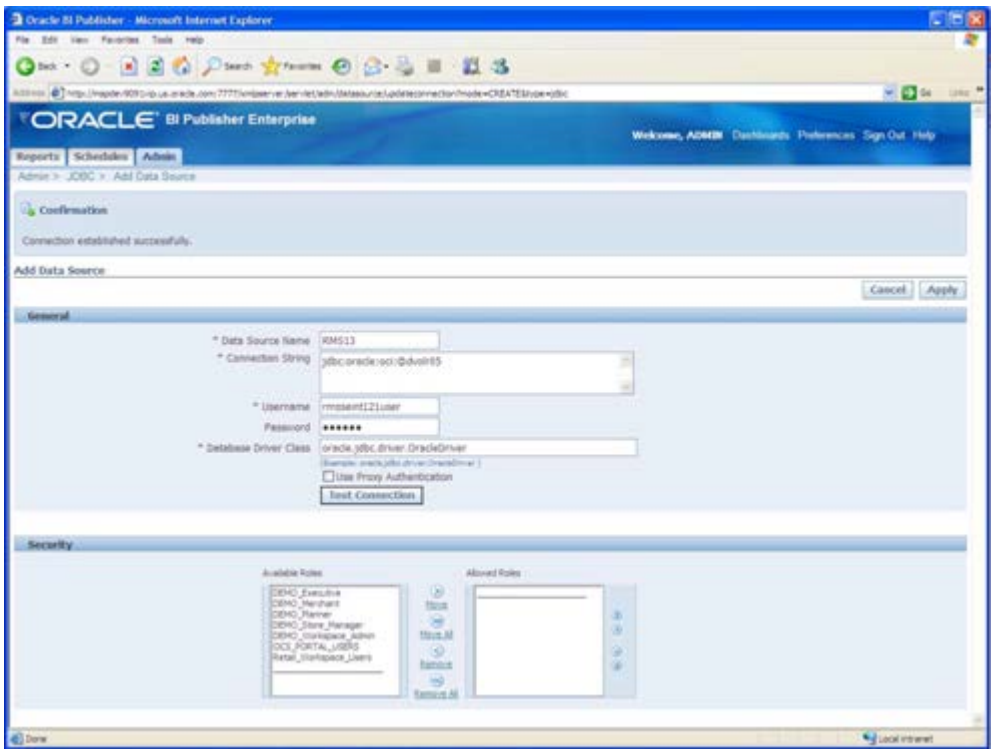
Prior to accessing reports in BI Publisher, you must create the BI Publisher JDBC connection for your database and verify that BI Publisher URLs are set correctly on your Oracle Application Server.

> **Note:** Oracle Retail recommends that retailers access the reports with the Guest user with no password. This is so RMS can access the reports without authentication. Guest with no password is a default user in BIP.

Setting up your JDBC Connection

Use the following procedure to create your JDBC connection for either a RAC database or a thin connection to a database.

1. Access your BIP repository.
   `/home/BIP/`
2. Log into BI Publisher as the administrator.
3. Under Data Sources, select JDBC Connections.
4. Create the BIP JDBC Connection.
   - If you are using JDBC for a RAC database, verify that the Data Source Name is set to RMS13.



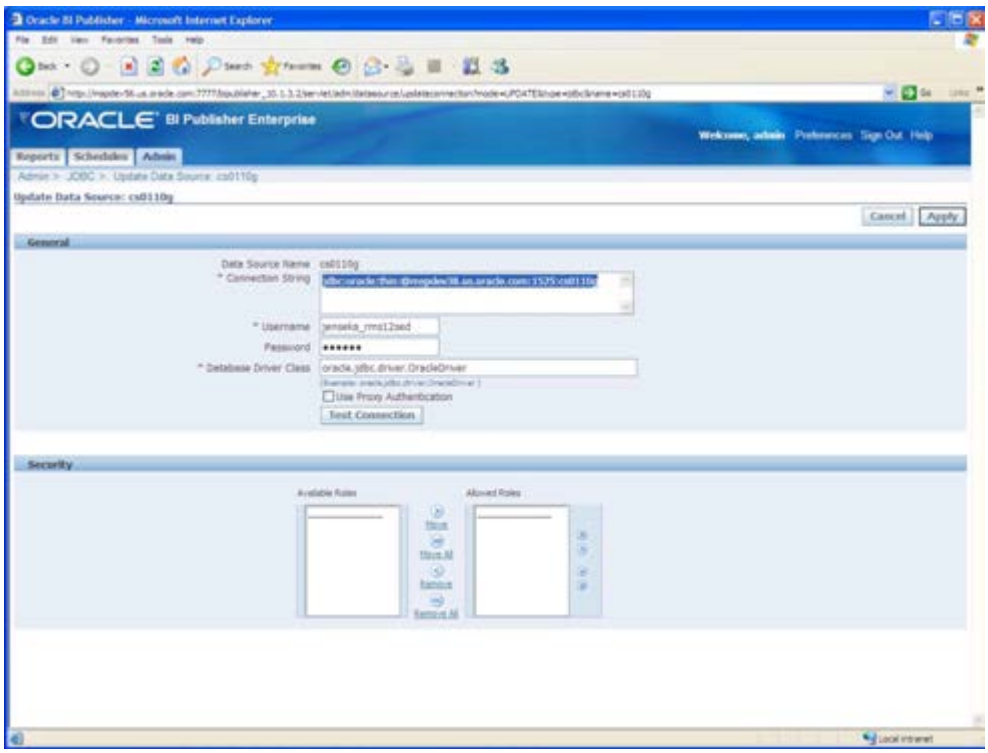
Example of a valid JDBC for a RAC database

- If you are using a thin connection to a database, the Connection String should look similar to the following:

  `jdbc:oracle:thin:@<DB_Server>:<DB_listener_port>:<$ORACLE_SID>`

  For example:

  `jdbc:oracle:thin:@mspdev38.us.oracle.com:1525:cs0110g`



Example of a thin connection to a database

5. Click Apply and close BI Publisher.

Verifying BI Publisher URLs

The BI Publisher URLs are set during the installation of RMS. If the URLs are incorrect, edit the RMS.env file using a text editor to correct the errors.

Use the following procedure to verify that your BI Publisher URLs are correct on your Oracle Application Server.

1. From your Oracle Application Server, access your RMS.env file. Your RMS.env file can be found in the following location:

   ```
   $ORACLE_HOME/forms/server/<DIR>/RMS*.env
   ```

2. Verify that the following lines in the RMS.env file correspond to your JDBC connection values:

   ```
   ORACLE_RMS_REPORTS_HOST=http://<server>:<port>/
   ORACLE_RMS_REPORTS_SERVER=<context root of BIP app>
   ORACLE_RMS_RWSERVER=<user>/<application>/<version>/
   ```

   For example:

   ```
   ORACLE_RMS_REPORTS_HOST=http://mspdev9091vip.us.oracle.com:7777/
   ORACLE_RMS_REPORTS_SERVER=xmlpserver
   ORACLE_RMS_RWSERVER=/Guest/RMS/12.1int/
   ```

3. Verify that the "RMS13" datasource is set correctly in the xdo files.

   In all reports, there is a <report>.xdo. Those xdo files have a similar line that should be set to "RMS13" out of the box.

   ```
   <report version="1.1" xmlns="http://xmlns.oracle.com/oxp/xmlp"
   defaultDataSourceRef="RMS13">
   ```

13

Using Oracle Wallet

RMS Batch programs can be run using wallet alias as the first parameter to the batch command line arguments. This is enabled to prevent the security concerns around exposing database user id and password while running the batch programs.

The wallet creation steps are described in the RMS Installation Guide. The wallet and wallet alias creation are a required in order to use batch programs in secured mode.

If we assume wallet alias is “dvols29_rms01batch”;

Usage:

```
./dtesys $UP
```

(where UP will be set during installation to the wallet alias i.e. $UP=/@dvols29_rms01batch)

or

```
./dtesys /@dvols29_rms01batch (wallet alias)
```