Oracle® Retail Merchandising System

Back-end Configuration and Operations Guide - Volume 3
Release 12.1

October 2008

ORACLE

Oracle® Reatil Merchandising System Back-end Configuration and Operations Guide - Volume 3, Release 12.1

Primary Author: Nathan Young

Value-Added Reseller (VAR) Language

Oracle Retail VAR Applications

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

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

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

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

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

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

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

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

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

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

Contents

Preface ..... **ix**
Audience ix
Related Documents ix
Customer Support ix
Review Patch Documentation x
Oracle Retail Documentation on the Oracle Technology Network x
Conventions x
1 Pro*C Restart and Recovery **1**
Table Descriptions and Definitions 1
restart_control 2
restart_program_status 3
restart_program_history 4
restart_bookmark 5
v_restart_x 6
Data Model Discussion 6
Why restart_program_status and restart_bookmark are Separate Tables 6
Physical Set-Up 6
Table and File-Based Restart/Recovery 7
API Functional Descriptions 9
restart_init 9
restart_file_init 10
restart_commit 10
restart_file_commit 10
restart_close 11
parse_array_args 11
restart_file_write 11
restart_cat 11
Restart Headers and Libraries 12
Updated restart headers and libraries 13
New Restart/Recovery Functions 14
Query-Based Commit Thresholds 16
2 Pro*C Multi-Threading **17**
Threading Description 17
Threading Function for Query-Based 18
Restart View for Query-Based 18
Thread Scheme Maintenance 20
File-Based 20
Query-Based 21
Batch Maintenance 21
Scheduling and Initialization of Restart Batch 22

Pre- and Post-Processing 22
3 Pro*C Array Processing 23
4 Pro*C Input and Output Formats 25
General Interface Discussion 25
Standard File Layouts 25
Detail Only Files 25
Master and Detail Files 26
Electronic Data Interchange (EDI) 28
5 RMS Internationalization and Localization 29
Key RMS Tables Related to Internationalization 29
FORM_ELEMENTS 29
FORM_ELEMENTS_LANGS 29
MENU_ELEMENTS 30
MENU_ELEMENTS_LANGS 30
FORM_MENU_LINK 30
CODE_DETAIL_TRANS 30
6 Custom Post Processing 31
7 RMS – Oracle E-Business Suite Integration 33
Architecture 34
Oracle E-Business Suite–Oracle Retail Outbound Process Flow 34
Oracle Retail–Oracle E-Business Suite Inbound Process Flow 36
RMS Business Processes Configuration 36
Organization Units 37
Currency Exchange Rates 37
Supplier Address Types 37
Country Codes 37
VAT Codes 37
RMS General Ledger Setup 37
Suppliers 37
Freight Terms, Payment Terms, Currency Exchange Rates, and General Ledger Chart of Accounts 37
Oracle E-Business Suite Org Units and Site IDs 38
System Parameter Maintenance 38
General Ledger Cross-Reference 38
General Ledger Account Maintenance 38
RMS Stock Ledger 38
ReSA Sales Information 39
Design Overview 39
Data Mapping 39
General Ledger Chart of Accounts Mapping 39
Supplier Mapping 41
Supplier Address Mapping 45

Currency Rates Mapping 47
Freight Terms Mapping 48
Payment Terms Mapping 49
Stock Ledger, Sales Journal, Miscellaneous Payments 50

Preface

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

Audience

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

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

Related Documents

For more information, see the following documents in the Oracle Retail Merchandising System Release 12.1 documentation:

- *Oracle Retail Merchandising System Data Model*
- *Oracle Retail Merchandising System Installation Guide*
- *Oracle Retail Merchandising System Release Notes*
- *Oracle Retail Merchandising System Online Help*
- *Oracle Retail Merchandising System Operations Guide – Volume 1*
- *Oracle Retail Merchandising System Operations Guide – Volume 2*
- *Oracle Retail Merchandising System Reports User Guide*
- *Oracle Retail Merchandising System User Guide*
- *Oracle Retail Merchandising Batch Schedule*
- *Oracle Retail Merchandising Data Conversion Operations Guide*
- *Oracle Retail Merchandising Implementation Guide*
- *Oracle Retail Sales Audit User Guide*
- *Oracle Retail Trade Management User Guide*

Customer Support

- https://metalink.oracle.com

When contacting Customer Support, please provide:

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

- Screen shots of each step you take.

Review Patch Documentation

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

Oracle Retail Documentation on the Oracle Technology Network

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

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

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

Conventions

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

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

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

A hyperlink appears like this.

1

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:

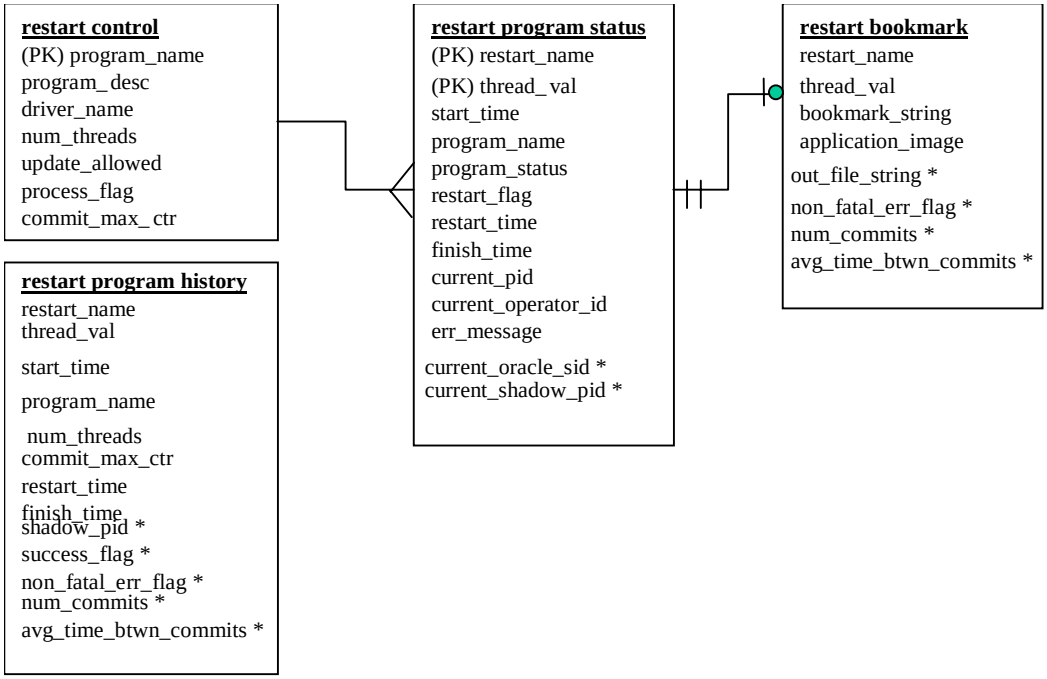
- Recover a halted process from the point of failure
- Prevent system halts due to large numbers of transactions
- Allow multiple instances of a given process to be active at the same time

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 diagram for the entity relationship diagram, followed by table descriptions.



Entity Relationship

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

Data Model Discussion

Why restart_program_status and restart_bookmark are Separate Tables

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 set up 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:

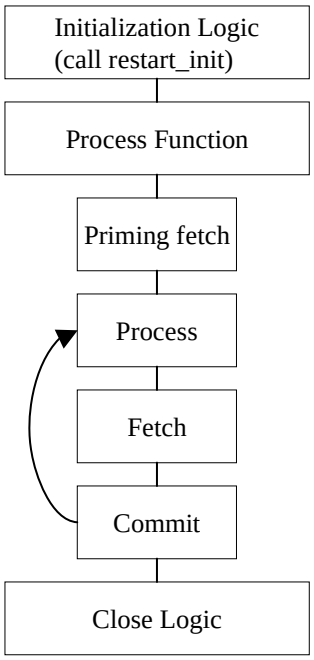
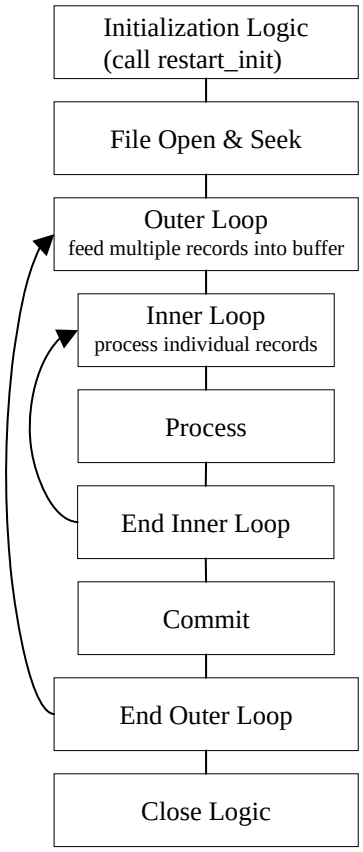


Table-Based Restart/Recovery Program Flow

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

The current RMS restart/recovery library has been updated in RMS versions 9, 10, 11, and 12 to enhance maintainability, enable easier coding and improve performance. While the current mechanism and functionality of batch restart/recovery are preserved, the following improvements and enhancements have been done:

- Organize global variables associated with restart recovery
- Allow the batch developer full control of restart recovery variables parameter passing during initialization
- Remove temporary write files to speed up the commit process
- Move more information and processing from the batch code into the library code
- Add 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.

2

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.

3

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.

4

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.

5

RMS Internationalization and Localization

The technical infrastructure of RMS supports languages other than English. The software can efficiently handle multiple languages. Tables have been added to RMS to accommodate internationalization. The client sets up the user's language preferences. RMS determines the user's language setting and displays the code string associated with it. RMS has a fail/safe mechanism built into the code. If the user's preference language string is not found, then RMS rolls back to English.

> **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)

Key RMS Tables Related to Internationalization

Several new tables were created to 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 DEFAULT_ACCESS_KEY field. At runtime, that character will be marked in the string, and function as the access key. Any time the retailer changes the DEFAULT_LABEL_PROMPT or DEFAULT_ACCESS_KEY, 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.

6

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.

7

RMS – Oracle E-Business Suite Integration

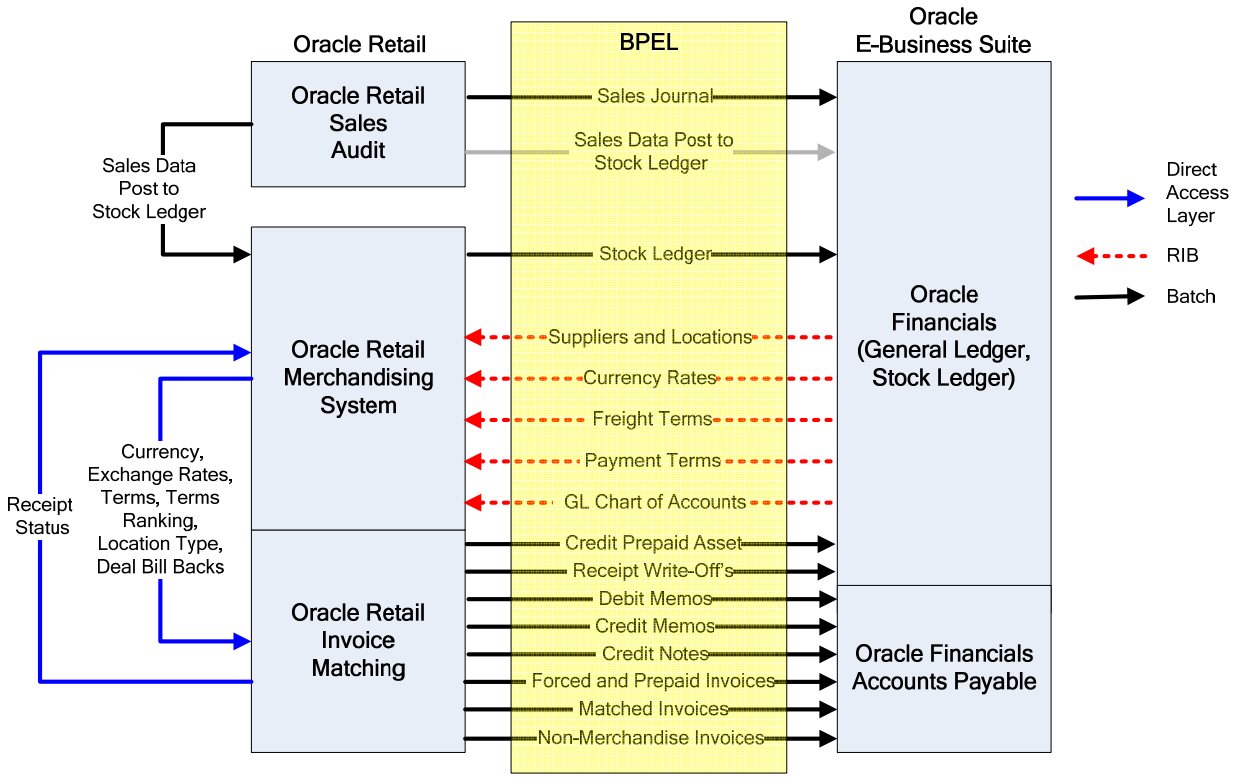
RMS, Oracle Retail Sales Audit (ReSA) and Oracle Retail Invoice Matching (ReIM) can be integrated with Oracle E-Business Suite 12.0.4. This integration allows RMS to obtain the following data from Oracle E-Business Suite:

- General ledger accounting data
- Supplier and supplier site data
- Currency rates
- Freight terms
- Payment terms

From Oracle Retail applications, inventory and sales accounting information can be sent to Oracle E-Business Suite.

The financial integration solution is delivered through middleware via a publish-subscribe model. Financial reference data is published from Oracle E-Business Suite (financial applications). Oracle Retail then subscribes to these processes through its own middleware architecture.

On the outbound transactional flows, the middleware populates financial staging tables. These staging tables are then polled and used to populate the standard financial open interface capabilities in Oracle General Ledger (GL) and Accounts Payable (AP).



Oracle Retail and Oracle Financials Integration—Logical View

Architecture

The integration solution is managed using these middleware components:

- Oracle BPEL (Business Process Execution Language) Process Manager
- Oracle Retail Integration Bus (RIB)

The applications integrated are as follows:

- Oracle General Ledger (GL)
- Oracle Accounts Payable (AP)
- Oracle Retail Merchandising System (RMS)
- Oracle Retail Invoice Matching (ReIM)
- Oracle Retail Sales Audit (ReSA)

Oracle BPEL Process Manager is a Java-based solution that runs on Oracle Application Server. Oracle BPEL Process Manager uses database adapters to connect to data sources in Oracle E-Business Suite.

RIB is an asynchronous message bus that subscribes to messages created and published by BPEL processes, to inject the message payload data elements into the RMS database. These Oracle Retail applications use Java Message Service (JMS) adapters to connect to RIB:

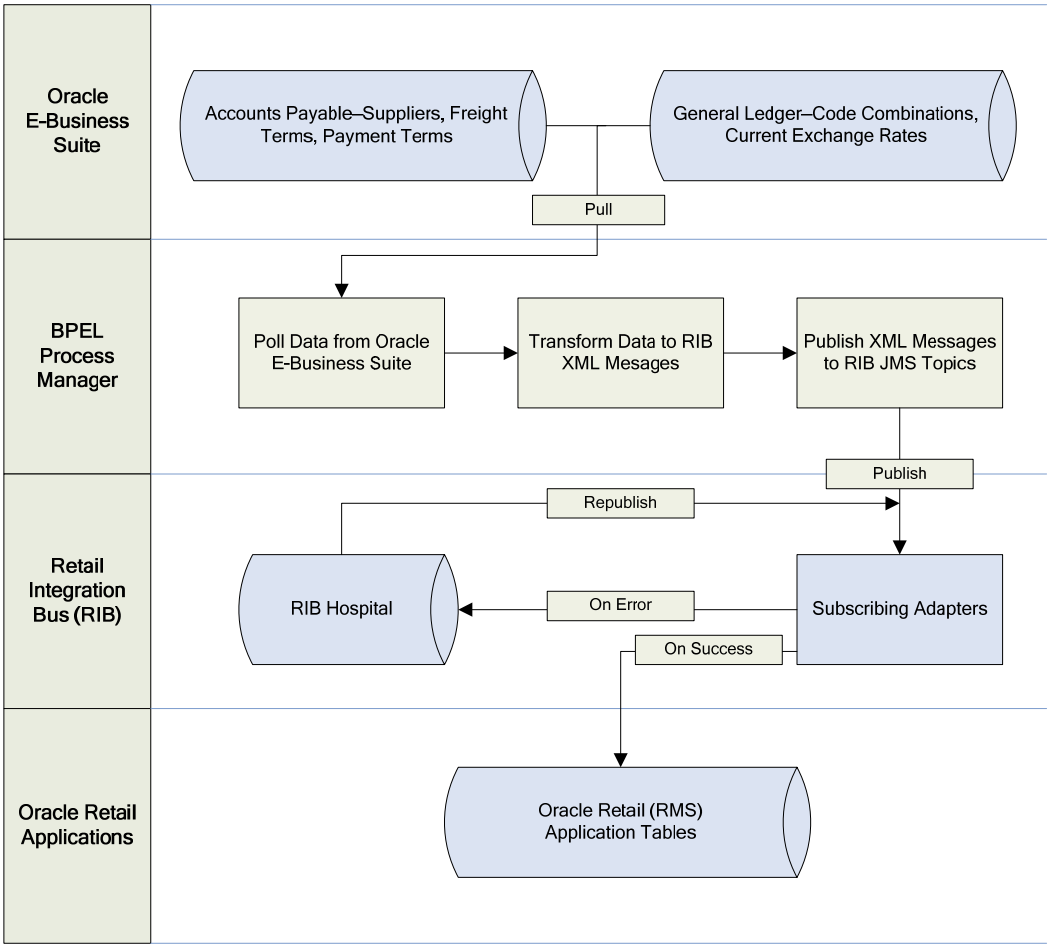
- Oracle Retail Management System (RMS)
- Oracle Retail Invoice Matching (ReIM)
- Oracle Retail Sales Audit (ReSA)

The BPEL and RIB middleware components are used to link and transform data as follows:

- Outbound data from Oracle E-Business Suite is sent on a net change basis to RIB. This data is processed within Oracle Retail applications, and the appropriate tables are updated.
- Summarized financial transactions from Oracle Retail applications are staged to financial tables. These tables are processed by BPEL processes to extract the data and populate the financial open interfaces in GL and AP. From that point, standard open interface processing is adopted.

Oracle E-Business Suite–Oracle Retail Outbound Process Flow

The following figure shows the outbound reference data flows that are supported for Oracle E-Business Suite through Oracle BPEL Process Manager.



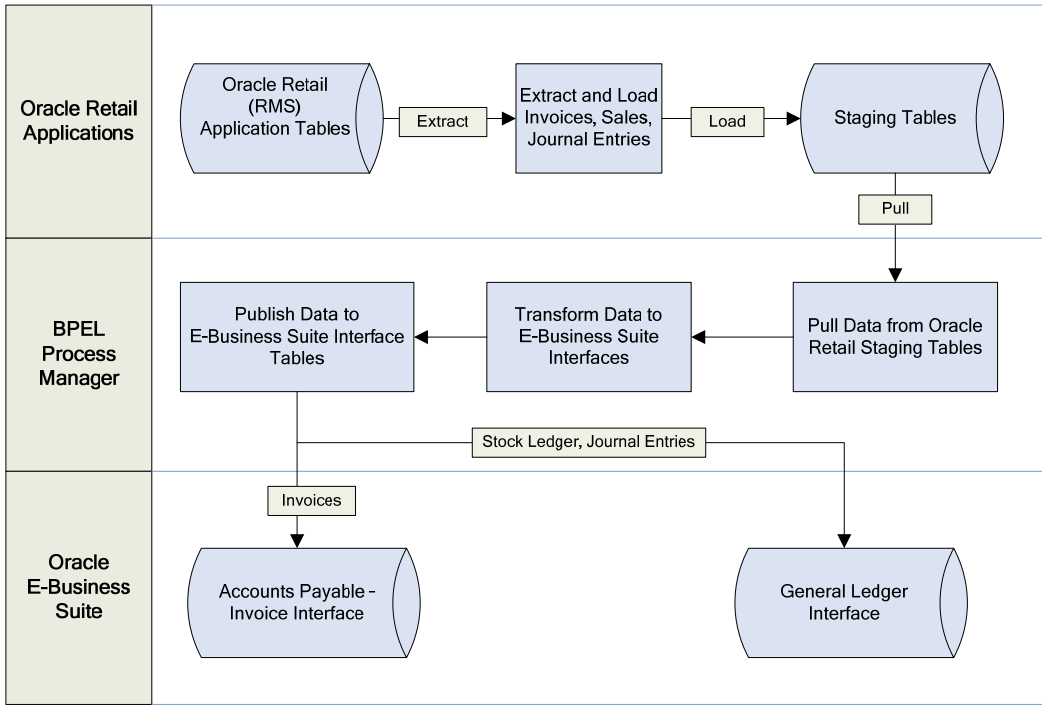
Outbound Process Flow

The outbound process flow is as follows:

1. From Oracle General Ledger account code combinations, suppliers, currency rates, freight terms, and payment terms are retrieved on a net change basis. All records added or changes are processed on a scheduled basis.
2. BPEL processes extract these records and perform data transformations to XML record formats that are compatible with RIB topics.
3. The XML messages are published to RIB. The subscribing RIB adapter attempts to update the appropriate business data object, based on message topic and action (create, modify, or delete). On success, the message is dropped from the JMS topic. On error, the message is written to RIB Error Hospital for manual review or automatic republication to the adapter.

Oracle Retail–Oracle E-Business Suite Inbound Process Flow

The following figure shows the inbound transactional data flows that are supported for Oracle E-Business Suite through Oracle BPEL Process Manager.



Inbound Process Flow

The inbound process flow is as follows:

1. The Oracle Retail business process extracts data objects to the standard financial integration staging tables. These tables are:
   - Matched purchase order receipts (new invoices)
   - Debit and credit memos
   - Inventory movements (stock ledger)
   - Sales audit (sales journal)
2. BPEL processes read the financial staging tables directly. These processes extract the details and perform data transformation to populate the standard GL or AP open interfaces.
3. After the standard open interface tables are populated, normal import and error management processes are performed in Oracle E-Business Suite.

RMS Business Processes Configuration

RMS System Options

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

- FINANCIAL_AP = O

  A value of the letter 'O' indicates that the financial system to which RMS is interfaced is Oracle E-Business Suite.

Organization Units

Use the Organizational Unit window 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 operating units are those that are associated with the set of books that is being used for the general ledger interface.

Currency Exchange Rates

Determine the Exchange Type being sent by Oracle (for example, 'Corporate') that you want RMS to use. Then use the Currency Exchange Type Mapping Window to map that External Exchange Type being sent by Oracle E-Business Suite to the RMS Exchange Type, 'Consolidation' (the RMS system option for Consolidation needs to be set).

Supplier Address Types

When defining valid address types in RMS, set the EXTERNAL_ADDR_IND to 'Y' for the order and remittance address types. For all other address types, leave this value as the default 'N'. Address types that are designated as 'External' do not allow maintenance through the RMS Supplier dialog.

Country Codes

When country codes are defined and seeded in RMS, only two character codes should be used, because Oracle E-Business Suite uses two character country codes by default.

VAT Codes

If a client is using both Oracle E-Business Suite and RMS, then the VAT codes defined in Oracle E-Business Suite need to observe the RMS limitations (that is, VAT codes cannot contain lower case characters and can be no more than six charaters in length).

RMS General Ledger Setup

The RMS table FIF_GL_SETUP holds the Oracle E-Business Suite set of books ID to which it links. This requires manual setup after the set of books is determined. The FIF_GL_SETUP table holds multiple records, one for each set of books.

For information about RMS's GL batch programs, see the General Ledger (GL) Batch chapter of the RMS Operations Guide Volume 1 - Batch Overviews and Designs for additional details.

Suppliers

Suppliers are created in Oracle E-Business Suite and exported to RMS. 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. (Functionality for partners' data is not affected.)

Freight Terms, Payment Terms, Currency Exchange Rates, and General Ledger Chart of Accounts

This data is created and updated in Oracle E-Business Suite and exported to RMS.

See the following sections of the RMS Operations Guide Volume 2 - Message Publication and Subscription Designs for additional details:

- Freight Terms Subscription API

- Payment Terms Subscription API
- Currency Exchange Rates Subscription API
- GL Chart of Accounts Subscription API

Oracle E-Business Suite Org Units and Site IDs

The data concepts of Org Units and Site IDs are used in RMS to mirror data maintained in Oracle E-Business Suite. RMS forms manage and view Oracle Org Units and Site IDs, and each store and warehouse is assigned to an Org Unit.

System Parameter Maintenance

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

General Ledger Cross-Reference

Navigate: RMS main menu > Finance> GL Cross Reference. The General Ledger Search window opens.

Associate general ledger accounts with Department, Location.

Where SYSTEM_OPTION_FINANCIAL_AP is 'O', disable debit and credit account IDs.

General Ledger Account Maintenance

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 'O', the form requires valid CC_IDs to be entered. The user can select segments and labels from the table FIF_GL_ACCT.

Turn off the debit and credit account CC_IDs.

RMS Stock Ledger

RMS tracks all inventory movements within the system through its stock ledger. Daily and period-based financial information transfers can be scheduled to Oracle General Ledger.

RMS allows three different levels to interface stock ledger information to Oracle General Ledger:

- Monthly
- Daily by subclass, class, or department
- Daily by transaction

The daily summarized interface and the monthly interface do not provide any access to detailed reference information.

For each interface (daily detail, daily summary, and monthly summary), specific transaction types can be included or excluded in the interface. This is determined in the general ledger cross-reference mapping. One interface, or a combination of interfaces, can be run to provide stock ledger data to Oracle General Ledger. Care must be taken to capture all necessary data, without duplicating any data.

This program summarizes stock ledger data from the monthly stock ledger table (MONTH_DATA), based on the level of information required, and writes the data to the financial general ledger staging table. The transactions extracted are determined by the code_type 'GLRT' (general ledger rolled transactions). The data written is then sent to the

financial applications for general ledger purposes. Stock ledger information can be rolled up at department, class, or subclass level. The level at which information is rolled up is determined by the gl_rollup field in the SYSTEM_OPTIONS table.

See the Stock Ledger Batch chapter of the RMS Operations Guide Volume 1 - Batch Overviews and Designs for additional details.

ReSA Sales Information

Oracle Retail Sales Audit (ReSA) allows summarized sales information to be forwarded to Oracle General Ledger for the sales journal. High-volume sales details can be aggregated by item, store and day, and transaction type. The retailer decides the aggregation level.

Interfacing data directly from ReSA to Oracle General Ledger is an optional process. Sales data from ReSA is also posted directly to the RMS stock ledger, and it can be interfaced to Oracle General Ledger through the stock ledger to general ledger interface. The decision to interface directly from ReSA to Oracle General Ledger is the retailer's decision. Setup steps are required to interface this data accurately to Oracle General Ledger without duplicating data.

Design Overview

The Oracle Retail SAEXPGL batch module posts all properly configured user-defined ReSA totals to Oracle General Ledger. Totals without errors are posted to the appropriate accounting ledger, as defined in the ReSA cross-reference user module. Depending on the ReSA system option Unit of Work, the data is sent at either the store/day or individual total level. Newly-revised totals that are already posted to the ledger have their previous revision reversed, and the new totals are posted to the appropriate accounts. Transactions from previous periods are posted to the current period.

See the Oracle Retail Sales Audit Batch chapter of the RMS Operations Guide Volume 1 - Batch Overviews and Designs for additional details.

Data Mapping

The following tables describe the data mappings between Oracle E-Business Suite and RMS.

General Ledger Chart of Accounts Mapping

The following table describes the mapping of chart of accounts data mapping from the Oracle E-Business Suite General Ledger Chart of Accounts to Oracle Retail Management System (RMS).

Field Passed from Oracle E-Business Suite	Oracle Retail RIB - GLCOADesc	API Create	API Update	Oracle Retail Database - FIF_GL_ACCT
code-combination-id	PRIMARY_ ACCOUNT	(Key)	(Key)	PRIMARY_ACCOUNT
segment1	ATTRIBUTE1	Pass	Pass	ATTRIBUTE1
segment2	ATTRIBUTE2	Pass	Pass	ATTRIBUTE2
segment3	ATTRIBUTE3	Pass	Pass	ATTRIBUTE3
segment4	ATTRIBUTE4	Pass	Pass	ATTRIBUTE4

Field Passed from Oracle E-Business Suite	Oracle Retail RIB - GLCOADesc	API Create	API Update	Oracle Retail Database - FIF_GL_ACCT
segment5	ATTRIBUTE5	Pass	Pass	ATTRIBUTE5
segment6	ATTRIBUTE6	Pass	Pass	ATTRIBUTE6
segment7	ATTRIBUTE7	Pass	Pass	ATTRIBUTE7
segment8	ATTRIBUTE8	Pass	Pass	ATTRIBUTE8
segment9	ATTRIBUTE9	Pass	Pass	ATTRIBUTE9
segment10	ATTRIBUTE10	Pass	Pass	ATTRIBUTE10
segment11	ATTRIBUTE11	Default	Ignore	ATTRIBUTE11
segment12	ATTRIBUTE12	Default	Ignore	ATTRIBUTE12
segment13	ATTRIBUTE13	Default	Ignore	ATTRIBUTE13
segment14	ATTRIBUTE14	Default	Ignore	ATTRIBUTE14
segment15	ATTRIBUTE15	Default	Ignore	ATTRIBUTE15
Segment_attribute 1	DESCRIPTION1	Default	Ignore	DESCRIPTION1
Segment_attribute 2	DESCRIPTION2	Default	Ignore	DESCRIPTION2
Segment_attribute 3	DESCRIPTION3	Default	Ignore	DESCRIPTION3
Segment_attribute 4	DESCRIPTION4	Default	Ignore	DESCRIPTION4
Segment_attribute 5	DESCRIPTION5	Default	Ignore	DESCRIPTION5
Segment_attribute 6	DESCRIPTION6	Default	Ignore	DESCRIPTION6
Segment_attribute 7	DESCRIPTION7	Default	Ignore	DESCRIPTION7
Segment_attribute 8	DESCRIPTION8	Default	Ignore	DESCRIPTION8
Segment_attribute 9	DESCRIPTION9	Default	Ignore	DESCRIPTION9
Segment_attribute 10	DESCRIPTION10	Default	Ignore	DESCRIPTION10
Segment_attribute 11	DESCRIPTION11	Default	Ignore	DESCRIPTION11
Segment_attribute 12	DESCRIPTION12	Default	Ignore	DESCRIPTION12
Segment_attribute 13	DESCRIPTION13	Default	Ignore	DESCRIPTION13
Segment_attribute 14	DESCRIPTION14	Default	Ignore	DESCRIPTION14
Segment_attribute 15	DESCRIPTION15	Default	Ignore	DESCRIPTION15
Chart_of_accounts_ID	SET_OF_BOOKS_ID	(Key)	(Key)	SET_OF_BOOKS_ID

Supplier Mapping

The following table describes the mapping of supplier data from Oracle E-Business Suite Accounts Payable to Oracle Retail Management System (RMS).

Field Passed from Oracle E-Business Suite	Oracle Retail RIB - VendorHdrDesc	API Create	API Update	Oracle Retail Database - SUPS	Window Edit (supvwedt)	Comments
ap_suppliers.segment1	SUPPLIER	(Key)	(Key)	SUPPLIER	(Key)	
ap_suppliers.vendor_name ap_supplier_sites_all.vendor_site_code	SUP_NAME	Pass	Pass	SUP_NAME	Disable	
NULL				SUP_NAME_SECONDARY		
NULL for Supplier. Mandatory for Supplier Site (ap_suppliers.segment1)				SUPPLIER_PARENT		
(One occurrence) ap_supplier_contacts.first_name ap_supplier_contacts.last_name	CONTACT_NAME	Pass	Ignore	CONTACT_NAME	Normal	Oracle E-Business Suite stores this data at either the supplier or supplier site level. One occurrence is sent, which is used in Oracle Retail for create only. After that, any updates are done by the user through the Oracle Retail window.
(One occurrence) Ap_supplier_contacts.area_code ap_supplier_contacts.phone	CONTACT_PHONE	Pass	Ignore	CONTACT_PHONE	Normal	See note for CONTACT_NAME.
(One occurrence) Ap_supplier_contacts.fax_area_code ap_supplier_contacts.fax	CONTACT_FAX	Pass	Ignore	CONTACT_FAX	Normal	See note for CONTACT_NAME.

Field Passed from Oracle E-Business Suite	Oracle Retail RIB - VendorHdrDesc	API Create	API Update	Oracle Retail Database - SUPS	Window Edit (supvwedt)	Comments
NULL	CONTACT_ PAGER	Default	Ignore	CONTACT_ PAGER	Normal	
(Derived) ap_suppliers	SUP_STATUS	(See note)	(See note)	SUP_STATUS	(See note)	For API: see information in the functional specification. For window: Disable this field unless the client uses VAT and current status is inactive. In that case, if the user tries to change the status to active, invoke the standard edit that the VAT region must be valued to change status to active.
Default: Y	QC_IND	Default	Ignore	QC_IND	Normal	
Default: 10	QC_PCT	Default	Ignore	QC_PCT	Normal	
Default: 1	QC_FREQ	Default	Ignore	QC_FREQ	Normal	
Default: N	VC_IND	Default	ignore	VC_IND	Normal	
Default: N	VC_PCT	Default	Ignore	VC_PCT	Normal	
	VC_FREQ	Default	Ignore	VC_FREQ	Normal	
(One occurrence) ap_supplier_ sites_all. invoice_ currency.code	CURRENCY_ CODE	Pass	Ignore	CURRENCY_ CODE	Normal	See note for CONTACT_NAME.
(One occurrence) ap_supplier_ sites_all. language	LANG	Pass	Ignore	LANG	Normal	See note for CONTACT_NAME.
(One occurrence) ap_supplier_ sites_all.terms_ id	TERMS	Pass	Ignore	TERMS	Normal	See note for CONTACT_NAME.

Field Passed from Oracle E-Business Suite	Oracle Retail RIB - VendorHdrDesc	API Create	API Update	Oracle Retail Database - SUPS	Window Edit (supvwedt)	Comments
(One occurrence) ap_supplier_sites_all.freight_terms_lookup_code	FREIGHT_TERMS	Pass	Ignore	FREIGHT_TERMS	Normal	See note for CONTACT_NAME.
Default: N	RET_ALLOW_IND	Default	Ignore	RET_ALLOW_IND	Normal	
Default: N	RET_AUTH_REQ	Default	Ignore	RET_AUTH_REQ	Normal	
---	RET_MIN_DOL_AMT	Default	Ignore	RET_MIN_DOL_AMT	Normal	
---	RET_COURIER	Default	Ignore	RET_COURIER	Normal	
---	HANDLING_PCT	Default	Ignore	HANDLING_PCT	Normal	
Default: N	EDI_PO_IND	Default	Ignore	EDI_PO_IND	Normal	
Default: N	EDI_PO_CHG	Default	Ignore	EDI_PO_CHG	Normal	
Default: N	EDI_PO_CONFIRM	Default	Ignore	EDI_PO_CONFIRM	Normal	
Default: N	EDI_ASN	Default	Ignore	EDI_ASN	Normal	
---	EDI_SALES_RPT_FREQ	Default	Ignore	EDI_SALES_RPT_FREQ	Normal	
Default: Y	EDI_SUPP_AVAILABLE_IND	Default	Ignore	EDI_SUPP_AVAILABLE_IND	Normal	
Default: N	EDI_CONTRACT_IND	Default	Ignore	EDI_CONTRACT_IND	Normal	
Default: N	EDI_INVC_IND	Default	Ignore	EDI_INVC_IND	Normal	
---		Default	N/A	EDI_CHANNEL_ID	Normal	
---	COST_CHG_PCT_VAR	Default	Ignore	COST_CHG_PCT_VAR	Normal	
---	COST_CHG_AMT_VAR	Default	Ignore	COST_CHG_AMT_VAR	Normal	
Default: N	REPLEN_APPROVAL_IND	Default	Ignore	REPLEN_APPROVAL_IND	Normal	
---	SHIP_METHOD	Default	Ignore	SHIP_METHOD	Normal	

Field Passed from Oracle E-Business Suite	Oracle Retail RIB - VendorHdrDesc	API Create	API Update	Oracle Retail Database - SUPS	Window Edit (supvwedt)	Comments
---	PAYMENT_ METHOD	Default	Ignore	PAYMENT_ METHOD	Normal	
---	CONTACT_ TELEX	Default	Ignore	CONTACT_ TELEX	Normal	
ap_supplier_ contacts.email_ address	CONTACT_ EMAIL	Pass	Ignore	CONTACT_ EMAIL	Normal	See note for CONTACT_NAME.
Default: E	SETTLEMENT_ CODE	Default	Ignore	SETTLEMENT_ CODE	Normal	
Default: N	PRE_MARK_ IND	Default	Ignore	PRE_MARK_ IND	Normal	
Default: N	AUTO_APPR_ INVC_IND	Default	Ignore	AUTO_APPR_ INVC_IND	Normal	
---	DBT_MEMO_ CODE	Default	Ignore	DBT_MEMO_ CODE	Normal	
Default: Y	FREIGHT_ CHARGE_IND	Default	Ignore	FREIGHT_ CHARGE_IND	Normal	
Default: N	AUTO_APPR_ DBT_MEMO_ IND	Default	Ignore	AUTO_APPR_ DBT_MEMO_ IND	Normal	
Default: N	PREPAY_INVC_ IND	Default	Ignore	PREPAY_INVC_ IND	Normal	
Default: N	BACKORDER_ IND	Default	Ignore	BACKORDER_ IND	Normal	
Default: 1000 (See note)	VAT_REGION	(See note)	Ignore	VAT_REGION	Normal	For API: see information in the functional specification.
---	INV_MGMT_ LVL	Default	Ignore	INV_MGMT_ LVL	Normal	
Default: N	SERVICE_PERF_ REQ_IND	Default	Ignore	SERVICE_PERF_ REQ_IND	Normal	
---	INVC_PAY_ LOC	Default	Ignore	INVC_PAY_ LOC	Normal	
---	INVC_ RECEIVE_LOC	Default	Ignore	INVC_ RECEIVE_LOC	Normal	
---	ADDINVC_ GROSS_NET	Default	Ignore	ADDINVC_ GROSS_NET	Normal	
Default: NDD	DELIVERY_ POLICY	Default	Ignore	DELIVERY_ POLICY	Normal	

Field Passed from Oracle E-Business Suite	Oracle Retail RIB - VendorHdrDesc	API Create	API Update	Oracle Retail Database - SUPS	Window Edit (supvwedt)	Comments
---	COMMENT_ DESC	Default	Ignore	COMMENT_ DESC	Normal	
---	DEFAULT_ ITEM_LEAD_ TIME	Default	Ignore	DEFAULT_ ITEM_LEAD_ TIME	Normal	
---	DUNS_ NUMBER	Default	Ignore	DUNS_ NUMBER	Normal	
---	DUNS_LOC	Default	Ignore	DUNS_LOC	Normal	
Default: N	BRACKET_ COSTING_IND	Default	Ignore	BRACKET_ COSTING_IND	Normal	
---	VMI_ORDER_ STATUS	Default	Ignore	VMI_ORDER_ STATUS	Normal	
Default: Y	DSD_ SUPPLIER_IND	Default	Ignore	DSD_IND	Normal	
---	END_DATE_ ACTIVE	Ignore	Ignore	N/A	N/A	
				SCALE_AIP_ORD ERS		

Supplier Address Mapping

The following table describes the mapping of address and site ID data from Oracle E-Business Suite Accounts Payable to Oracle Retail Management System (RMS).

Note: Records in Oracle E-Business Suite are sent to Oracle Retail applications when the following fields are populated in Oracle E-Business Suite: address_line1, city, and zip.

Field Passed from Oracle E-Business Suite	Oracle Retail RIB - VendorAddrDesc	API Create	API Update	Oracle Retail Database - ADDR	Window Edit (addr)	Comments
ap_supplier_sites _all.vendor_site_ id		(Key)	(Key)	ADDR_KEY	N/A	
(literal)	MODULE	(Key)	(Key)	MODULE	N/A	This should always be the literal 'SUPP'.
ap_supplier. segment1	KEY_VALUE_1	(Key)	(Key)	KEY_VALUE_1	N/A	This is the supplier number.
Null	KEY_VALUE_2	Default	Ignore	KEY_VALUE_2	N/A	
Default: 1	SEQ_NO	Generate	Ignore	SEQ_NO	N/A	

Field Passed from Oracle E-Business Suite	Oracle Retail RIB - VendorAddrDesc	API Create	API Update	Oracle Retail Database - ADDR	Window Edit (addr)	Comments
(Derived) ap_supplier_sites_all.purchasing_site_flag ap_supplier_sites_all.pay_site_flag	ADDR_TYPE	(See note)	(See note)	ADDR_TYPE	N/A	
Default: N	PRIMARY_ADDR_IND	Generate	Generate	PRIMARY_ADDR_IND	Normal	Logic in API determines.
ap_supplier_sites_all.address_line1	ADD_1	Pass	Pass	ADD_1	Disable	
ap_supplier_sites_all.address_line2	ADD_2	Pass	Pass	ADD_2	Disable	
ap_supplier_sites_all.address_line3	ADD_3	Pass	Pass	ADD_3	Disable	
ap_supplier_sites_all.city	CITY	Pass	Pass	CITY	Disable	
ap_supplier_sites_all.state	STATE	Pass	Pass	STATE	Disable	
ap_supplier_sites_all.country	COUNTRY_ID	Pass	Pass	COUNTRY_ID	Disable	
ap_supplier_sites_all.zip	POST	Pass	Pass	POST	Disable	
(One occurrence) ap_supplier_contacts.first_name ap_supplier_contacts..last_name	CONTACT_NAME	Pass	Ignore	CONTACT_NAME	Normal	Oracle E-Business Suite passes the first contact name for each address created.
(One occurrence) ap_supplier_contacts.area_code ap_supplier_contacts.phone	CONTACT_PHONE	Pass	Ignore	CONTACT_PHONE	Normal	Oracle E-Business Suite passes the first contact name for each address created.
Null	CONTACT_TELEX	Default	Ignore	CONTACT_TELEX	Normal	

Field Passed from Oracle E-Business Suite	Oracle Retail RIB - VendorAddrDesc	API Create	API Update	Oracle Retail Database - ADDR	Window Edit (addr)	Comments
(One occurrence) ap_supplier_contacts.fax_area_code ap_supplier_contacts.fax	CONTACT_FAX	Pass	Ignore	CONTACT_FAX	Normal	Oracle E-Business Suite passes the first contact name for each address created.
(One occurrence) ap_supplier_contacts.email_address	CONTACT_EMAIL	Pass	Ignore	CONTACT_EMAIL	Normal	Oracle E-Business Suite passes the first contact name for each address created.
N/A	N/A	Default	N/A	EDI_ADDR_CHG	N/A	
N/A	N/A	Default	N/A	COUNTY	N/A	
N/A	N/A	Default	N/A	PUBLISH_IND	N/A	
Multiple occurrences of information at lower level:						
ORG_ID	ORACLE_ORG_UNIT_ID	Pass	Pass	ORACLE_ORG_UNIT_ID	N/A	

Currency Rates Mapping

The following table describes the mapping of currency rates data from Oracle E-Business Suite to Oracle Retail Management System (RMS).

Field Passed from Oracle E-Business Suite	Oracle Retail RIB - CurrRateDesc	API Create	API Update	Oracle Retail Database - Currency Rates	Window Edit - Currency Exchange Rates	Comments
from-currency	FROM_CURRENCY	(Key)	(Key)	CURRENCY_CODE	Allow view only	
conversion-date	CONVERSION_DATE	(Key)	(Key)	EFFECTIVE_DATE	Allow view only	
conversion-type	USER_CONVERSION_TYPE	Translate and use for key	Translate and use for key	EXCHANGE_TYPE	Allow view only	Read the FIF-CURRENCY-XREF table to translate incoming value to rms-exchange-type.

Field Passed from Oracle E-Business Suite	Oracle Retail RIB - CurrRateDesc	API Create	API Update	Oracle Retail Database - Currency Rates	Window Edit - Currency Exchange Rates	Comments
conversion-rate	CONVERSION_RATE	Pass	Pass	EXCHANGE_RATE	Allow view only	
to-currency	TO_CURRENCY	Use for filter	Use for filter	N/A	N/A	Check this value against the currency-code field in the system-options table. If they do not match, ignore this record.

Freight Terms Mapping

The following table describes mapping of freight terms data from Oracle E-Business Suite to Oracle Retail Management System (RMS).

Field Passed from Oracle E-Business Suite	Oracle Retail RIB - FrtTermDesc	API Create	API Update	Oracle Retail Database - FREIGHT_TERMS
lookup-code	FREIGHT_TERMS	(Key)	(Key)	FREIGHT_TERMS
meaning	TERM_DESC	Pass	Pass	TERM_DESC
start_date_active	START_DATE_ACTIVE	Pass	Pass	START_DATE_ACTIVE
end_date_active	END_DATE_ACTIVE	Pass	Pass	END_DATE_ACTIVE
enabled-flag	ENABLED_FLAG	Pass	Pass	ENABLED_FLAG

Payment Terms Mapping

The following table describes mapping of payment terms data from Oracle E-Business Suite to Oracle Retail Management System (RMS).

Note: In the table, an asterisk (*) indicates that this field is maintained at header level in Oracle E-Business Suite.

Field Passed from Oracle E-Business Suite AP_TERMS_TL(AT) AP_TERMS_LINES (ATL)	**Oracle Retail RIB - PayTermDesc?**	**Oracle Retail RIB - PayTermDtl?**	**API Create**	**API Update**	**Oracle Retail Database TERMS_HEADER**
Oracle Retail Database TERMS_HEAD					
AT.Term_Id	TERMS		(Key)	(Key)	TERMS
AT.Name	TERMS_CODE		Pass	Pass	TERMS_CODE
AT.Description	TERMS_DESC		Pass	Pass	TERMS_DESC
AT.Rank	RANK		Pass	pass	RANK
Default NULL	DUE_DAYS				
Derived (AT)	ENABLED_ FLAG				
AT.Start_Date_Active	START_DATE_ ACTIVE				
AT.End_Date_Active	END_DATE_ ACTIVE				
Default NULL	DISCDAYS				
Default NULL	DISCDAYS				
Oracle Retail Database TERMS_DETAIL					
AT.Term-id	TERMS		(Key)	(Key)	TERMS
ATL.sequence-num		TERMS_SEQ	(Key)	(Key)	TERMS_SEQ
ATL.due-days		DUE_DAYS	Pass	Pass	DUEDAYS
ATL.due-amount		DUE_MAX_ AMOUNT	Pass	Pass	DUE_MAX_ AMOUNT
due-day-of-month		DUE_DOM	Pass	Pass	DUE_DOM
ATL.due-months-forward		DUE_MM_FWD	Pass	Pass	DUE_MM_FWD
ATL.discount-days		DISCDAYS	Pass	Pass	DISCDAYS

Field Passed from Oracle E-Business Suite AP_TERMS_TL(AT) AP_TERMS_LINES (ATL)	Oracle Retail RIB - PayTermDesc?	Oracle Retail RIB - PayTermDtl?	API Create	API Update	Oracle Retail Database TERMS_HEADER
ATL.discount-percent		PERCENT	Pass	Pass	PERCENT
ATL.discount-day-of-month		DISC_DOM	Pass	Pass	DISC_DOM
ATL.discount-months-forward		DISC_MM_FWD	Pass	Pass	DISC_MM_FWD
ATL.fixed-date		FIXED_DATE	Pass	Pass	FIXED_DATE
Derived (ATL)		ENABLED_FLAG	Pass	Pass	ENABLED_FLAG
ATL.start-date-active*		START_DATE_ ACTIVE	Pass	Pass	START_DATE_ ACTIVE
ATL.end-date-active*		END_DATE_ ACTIVE	Pass	Pass	END_DATE_ACTIVE
ATL.due-cutoff-day*		CUTOFF_DAY	Pass	Pass	CUTOFF_DAY

Stock Ledger, Sales Journal, Miscellaneous Payments

The following table describes the mapping of Oracle Retail stock ledger, sales journal, and miscellaneous payments data to Oracle E-Business Suite.

Column Name	Data Type	Column Name	Data Type	Req'd	Remarks
STATUS	VARCHAR2(50)	STATUS	VARCHAR2(50)	Yes	Straight mapping.
SET_OF_BOOKS_ ID	NUMBER(15)	SET_OF_BOOKS_ ID	NUMBER(15)	Yes	Straight mapping. Oracle Retail sends a valid value.
ACCOUNTING_ DATE	DATE	ACCOUNTING_ DATE	DATE	Yes	Straight mapping.
CURRENCY_ CODE	VARCHAR2(15)	CURRENCY_ CODE	VARCHAR2(15)	Yes	Straight mapping. Oracle Retail synchronizes reference data with Oracle E-Business Suite.
DATE_CREATED	DATE	DATE_ CREATED	DATE	Yes	Straight mapping.
CREATED_BY	NUMBER(15)	CREATED_BY	NUMBER(15)	Yes	
ACTUAL_FLAG	VARCHAR2(1)	ACTUAL_FLAG	VARCHAR2(1)	Yes	Straight mapping.
USER_JE_ CATEGORY_ NAME	VARCHAR2(25)	USER_JE_ CATEGORY_ NAME	VARCHAR2(25)	Yes	Straight mapping. Oracle Retail sends a valid value.
USER_JE_ SOURCE_NAME	VARCHAR2(25)	USER_JE_ SOURCE_NAME	VARCHAR2(25)	Yes	Straight mapping. Oracle Retail sends a valid value.

Column Name	Data Type	Column Name	Data Type	Req'd	Remarks
CURRENCY_ CONVERSION_ DATE	DATE	CURRENCY_ CONVERSION_ DATE	DATE		Straight mapping.
CURRENCY_ CONVERSION_ TYPE	VARCHAR2(30)	USER_ CURRENCY_ CONVERSION_ TYPE	VARCHAR2(30)		The only valid values are 'Corporate' and 'Spot'.
		CURRENCY_ CONVERSION_ RATE	NUMBER		Not sent by Oracle Retail. Conversion rate is derived from the GL_DAILY_RATES table based on conversion date, conversion type, currency_code, and functional currency.
ACCT_ SEGMENT1	VARCHAR2(25)	SEGMENT1	VARCHAR2(25)		Not sent by Oracle Retail because the code_combination_id is sent.
ACCT_ SEGMENT2	VARCHAR2(25)	SEGMENT2	VARCHAR2(25)		Not sent by Oracle Retail because the code_combination_id is sent.
ACCT_ SEGMENT3	VARCHAR2(25)	SEGMENT3	VARCHAR2(25)		Not sent by Oracle Retail because the code_combination_id is sent.
ACCT_ SEGMENT4	VARCHAR2(25)	SEGMENT4	VARCHAR2(25)		Not sent by Oracle Retail because the code_combination_id is sent.
ACCT_ SEGMENT5	VARCHAR2(25)	SEGMENT5	VARCHAR2(25)		Not sent by Oracle Retail because the code_combination_id is sent.
ACCT_ SEGMENT6	VARCHAR2(25)	SEGMENT6	VARCHAR2(25)		Not sent by Oracle Retail because the code_combination_id is sent.
ACCT_ SEGMENT7	VARCHAR2(25)	SEGMENT7	VARCHAR2(25)		Not sent by Oracle Retail because the code_combination_id is sent.
ACCT_ SEGMENT8	VARCHAR2(25)	SEGMENT8	VARCHAR2(25)		Not sent by Oracle Retail because the code_combination_id is sent.

Column Name	Data Type	Column Name	Data Type	Req'd	Remarks
ACCT_ SEGMENT9	VARCHAR2(25)	SEGMENT9	VARCHAR2(25)		Not sent by Oracle Retail because the code_combination_id is sent.
ACCT_ SEGMENT10	VARCHAR2(25)	SEGMENT10	VARCHAR2(25)		Not sent by Oracle Retail because the code_combination_id is sent.
ENTERED_DR_ AMOUNT	NUMBER(20,4)	ENTERED_DR	NUMBER		Straight mapping and truncation per Oracle Retail column sizing constraints.
ENTERED_CR_ AMOUNT	NUMBER(20,4)	ENTERED_CR	NUMBER		Straight mapping and truncation per Oracle Retail column sizing constraints.
		ACCOUNTED_ DR	NUMBER		Not sent. This is derived during GL import from the GL Daily Rates table.
		ACCOUNTED_ CR	NUMBER		Not sent. This is derived during GL import from the GL Daily Rates table.
TRANSACTION_ DATE	DATE	TRANSACTION_ DATE	DATE		Straight mapping.
REFERENCE1	VARCHAR2(20)	REFERENCE21	VARCHAR2(100)		Straight mapping.
REFERENCE2	VARCHAR2(20)	REFERENCE22	VARCHAR2(240)		Straight mapping.
REFERENCE3	VARCHAR2(20)	REFERENCE23	VARCHAR2(100)		Straight mapping.
REFERENCE4	VARCHAR2(20)	REFERENCE24	VARCHAR2(100)		Straight mapping.
REFERENCE5	VARCHAR2(20)	REFERENCE25	VARCHAR2(240)		Straight mapping.
		JE_BATCH_ID	NUMBER(15)		Do not map.
PERIOD_NAME	VARCHAR2(15	PERIOD_NAME	VARCHAR2(15)		Straight mapping. The period names are manually synchronized.
		JE_HEADER_ID	NUMBER(15)		Leave null. For internal use only.
		JE_LINE_NUM	NUMBER(15)		Do not map.
		CHART_OF_ ACCOUNTS_ID	NUMBER(15)		Do not map.
		FUNCTIONAL_ CURRENCY_ CODE	VARCHAR2(15)		Do not map.
CODE_ COMBINATION_ ID	NUMBER(15)	CODE_ COMBINATION_ ID	NUMBER(15)		Straight mapping.

Column Name	Data Type	Column Name	Data Type	Req'd	Remarks
ATTRIBUTE1	VARCHAR2(20)	ATTRIBUTE1	VARCHAR2(150)		Do not pull these columns.
ATTRIBUTE2	VARCHAR2(20)	ATTRIBUTE2	VARCHAR2(150)		Do not pull these columns.
		GL_SL_LINK_ID	NUMBER		Leave these null.
		GL_SL_LINK_ TABLE	VARCHAR2(30)		Leave these null.
ATTRIBUTE3	VARCHAR2(20)	ATTRIBUTE3	VARCHAR2(150)		Do not pull these columns.
ATTRIBUTE4	VARCHAR2(20)	ATTRIBUTE4	VARCHAR2(150)		Do not pull these columns.
ATTRIBUTE5	VARCHAR2(20)	ATTRIBUTE5	VARCHAR2(150)		Do not pull these columns.
ATTRIBUTE6	VARCHAR2(20)	ATTRIBUTE6	VARCHAR2(150)		Do not pull these columns.
		CONTEXT	VARCHAR2(150)		Do not pull these columns.
		CONTEXT2	VARCHAR2(150)		Do not pull these columns.
		INVOICE_DATE	DATE		Do not map.
		TAX_CODE	VARCHAR2(15)		Do not map.
		INVOICE_ IDENTIFIER	VARCHAR2(20)		Do not map.
		INVOICE_ AMOUNT	NUMBER		Do not map.
		CONTEXT3	VARCHAR2(150)		Do not map.
		USSGL_ TRANSACTION _CODE	VARCHAR2(30)		Do not map.
		DESCR_FLEX_ ERROR_ MESSAGE	VARCHAR2(240)		Do not map.
		JGZZ_RECON_ REF	VARCHAR2(240)		Do not map.
		REFERENCE_ DATE	DATE		Do not map.
PGM_NAME	VARCHAR2(100)				Do not map.
		ENCUMBRANCE_ TYPE_ID	NUMBER		Do not map.
		BUDGET_ VERSION_ID	NUMBER		Do not map.

Column Name	Data Type	Column Name	Data Type	Req'd	Remarks
		AVERAGE_ JOURNAL_ FLAG	VARCHAR2(1)		Do not map.
		ORIGINATING_ BAL_SEG_ VALUE	VARCHAR2		Do not map.
		SEGMENT11-30	VARCHAR2(25)		Do not map.
		REFERENCE1,3,4,6 -9	VARCHAR2(100)		Do not map.
		REFERENCE10,2,5	VARCHAR2(240)		Do not map.
		REFERENCE11-20	VARCHAR2(100)		Do not map.
		REFERENCE26-30	VARCHAR2(240)		Do not map.
		DATE_CREATED_ IN_GL	DATE		Do not map.
		WARNING_ CODE	VARCHAR2(4)		Do not map.
		STATUS_ DESCRIPTION	VARCHAR2(240)		Do not map.
		STAT_AMOUNT	NUMBER		Do not map.
		GROUP_ID	NUMBER(15)		Do not map.
		REQUEST_ID	NUMBER(15)		Do not map.
		SUBLEDGER_ DOC_ SEQUENCE_ID	NUMBER		Do not map.
		SUBLEDGER_ DOC_SEQUENCE_ VALUE	NUMBER		Do not map.