Oracle® Retail Merchandising System

Operations Guide Addendum

Release 10.1.17

October 2006

Oracle® Merchandising System Operations Guide Addendum, Release 10.1.17

Primary Author: Nathan Young

Contents

Preface

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

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

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

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

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

Related Documents

For more information, see the following documents in the Oracle Retail Merchandising System Release 10.1.17 documentation set:

- Oracle Retail Merchandising System Release Notes
- Oracle Retail Merchandising System Installation Guide
- Oracle Retail Merchandising System Batch Schedule

Customer Support

- https://metalink.oracle.com

When contacting Customer Support, please provide:

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

Conventions

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

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

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

A hyperlink appears like this.

Introduction

The information in this document reflects modifications and updates to the latest Oracle Retail Merchandising System Operations Guide. Using this document in conjunction with that guide provides retailers with a complete overview of the application.

For more specific information regarding enhancements and modifications made to the previous Oracle Retail Merchandising System release, see the *Oracle Retail Merchandising System 10.1.17 Release Notes*.

1 Batch Designs

Sales Audit Get Reference (sagetref)

Design Overview

This RMS program will fetch all reference information needed by saimplog.pc and write the information out to separate output files.

One file contains a listing of all items in the system.

A second file contains information about all items that have wastage associated with them. A third file contains reference items.

- A fourth file contains primary variant information.
- A fifth file contains all variable weight UPC definitions in the system.
- A sixth file contains all of the valid store/day combinations in the system.
- A seventh file contains all code types and codes used in field level validation.
- An eighth file contains all error codes, error descriptions, and systems affected by the error.
- A ninth file contains the credit card validation mappings.
- A tenth file contains the store_pos mappings.
- An eleventh file contains the tender type mappings.
- A twelfth file contains the merchant code mappings.
- A thirteenth file contains the partner mappings.
- A fourteenth file contains the supplier mappings.
- A fifteenth file contains employee mappings.
- Finally, a sixteenth file contains promotion information.

These files are used by the automated audit to validate information without repeatedly hitting the database.

The tables affected include:

Table	Index	Select	Insert	Update	Delete
ITEM_MASTER	No	Yes	No	No	No
ITEM_LOC	No	Yes	No	No	No
VAR_UPC_EAN	No	Yes	No	No	No
SA_IMPORT_LOG	No	Yes	No	No	No
CURRENCIES	No	Yes	No	No	No
STORE_DAY	No	Yes	No	No	No
STORE	No	Yes	No	No	No
SA_STORE_DAY	No	Yes	No	No	No
CODE_DETAIL	No	Yes	No	No	No
SA_ERROR_CODES	No	Yes	No	No	No
SA_CC_VAL	No	Yes	No	No	No
SA_STORE_POS	No	Yes	No	No	No
POS_TENDER_TYPE_HEAD	No	Yes	No	No	No
NON_MERCH_CODE_HEAD	No	Yes	No	No	No
PARTNER	No	Yes	No	No	No
SUPS	No	Yes	No	No	No
SA_STORE_EMP	No	Yes	No	No	No
PROMHEAD	No	Yes	No	No	No
PROM_MIX_MATCH_HEAD	No	Yes	No	No	No
PROM_THRESHOLD_DEPT	No	Yes	No	No	No
PROM_THRESHOLD_SKU	No	Yes	No	No	No
CHANNELS	No	Yes	No	No	No

Stored Procedures/Shared Modules (Maintainability)

N/A

Function Level Description

main()

The standard Oracle Retail Pro*C main function that calls init(), process(), and final().

init()

This function initializes the necessary restart recovery variables. It calls the function retek_init().

process()

This function calls process_item_master_info() to retrieve item information from the database and writes it to a item and waste output file.

This function then calls:

- process_ref_item_info() to retrieve reference item information from the database and writes it to the reference item output file.
- process_prim_variant_info() to retrieve primary variant information from the database and writes it to a primary variant output file.
- process_var_upc_ean_info() to retrieve all variable weight UPC mappings from the database and writes it to the variable weight UPC output file.
- process_store_day_info() to retrieve all valid store day combinations from the database and writes it to the store day output file.
- process_codes_info() to retrieve all codes from the database that are used in file validation and writes it to the codes output file.
- process_error_info() to retrieve all errors from the database that are used in file validation and writes it to the error output file.
- process_cc_info() to retrieve all credit card validation mappings from the database and writes it to the credit card validation output file.
- process_store_pos_info() to retrieve all store/pos mappings from the database and writes it to the store POS output file.
- process_tender_type_info() to retrieve all tender type mappings from the database and writes it to the tender type output file.
- process_merch_codes_info() to retrieve all merchant code mappings from the database and writes it to the merchant code output file.
- process_partner_info() to retrieve all partner mappings from the database and writes it to the partner output file.
- process_supplier_info() to retrieve all supplier mappings from the database and writes it to the supplier output file.
- process_employee_info() to retrieve all employee mappings from the database and writes it to the employee output file.
- Finally, this function calls process_prom_info() to retrieve promotion information from the database and writes it to the promotion output file.

process_item_master_info()

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

size_item_master_arrays()

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

write_item_data()

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

write_waste_data()

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

process_ref_item_info()

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

size_ref_item_arrays()

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

write_ref_item_data()

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

process_prim_variant_info()

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

size_prim_variant_arrays()

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

write_prim_variant_data()

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

process_var_upc_ean_info()

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

size_var_upc_ean_arrays()

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

write_var_upc_data()

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

process_store_day_info()

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

size_store_day_arrays()

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

write_store_day_data()

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

process_codes_info()

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

size_codes_arrays()

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

write_codes_data()

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

process_error_info()

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

size_error_arrays()

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

write_error_data()

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

process_cc_val_info()

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

size_cc_val_arrays()

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

write_cc_val_data()

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

process_store_pos_info()

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

size_store_pos_arrays()

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

write_store_pos_data()

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

process_tender_type_info()

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

size_tender_type_arrays()

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

write_tender_type_data()

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

process_merch_codes_info()

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

size_merch_codes_arrays()

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

write_merch_codes_data()

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

process_partner_info()

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

size_partner_arrays()

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

write_partner_data()

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

process_supplier_info()

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

size_supplier_arrays()

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

write_supplier_data()

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

process_employee_info()

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

size_employee_arrays()

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

write_employee_data()

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

Process_prom_info()

This function queriesy promotion information from the promotion tables and uses this information to populate the promotion array. This function also calls size_prom_arrays() to allocate memory for the promotion array. The column selected is promotion. The information should be ordered by promotion. All records in the promotion array will be written to the promotion output file by calling write_prom_data().

size_prom_arrays()

This function allocates memory for the promotion array used in process_prom_info.

write_prom_data()

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

final()

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

Input Specifications

N/A

Output Specifications

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

Character field types are left-justified and blank padded.

Number field types are right-justified and zero padded.

Record Name	Field Name	Field Type	Default Value	Description
Item Data	Item	char(25)		Unique item identifier
	Dept	char(4)		Department identifier
	Class	char(4)		Class identifier
	Subclass	char(4)		Subclass identifier
	Standard_uom	char(4)		Standard UOM
Waste Data	Item	char(25)		Unique item identifier
	Waste_type	char(6)		Waste type identifier
	Waste_pct	number(16)		Waste percent
Ref Item Data	Item_parent	char(25)		Item Parent
	Item	char(25)		Unique item identifier
Primary Variant Data	Item_loc	number(10)		Item location
	Item	char(25)		Unique item identifier
	Primary_variant	char(25)		Primary variant
Variable UPC Data	Format_id	char(1)		Format identifier
	Format_desc	char(20)		Format description
	Prefix_length	number(1)		Prefix length
	Begin_item_digit	number(2)		Determines the first digit of the item number.
	Begin_var_digit	number(2)		Determines the first digit of the variable weight/price.
	Check_digit	number(2)		Position of the check digit.
	Prefix	number(2)		Item master prefix
Store Day Data	Store	number(10)		Store number
	Business_date	char(8)		Business date – format: YYYYMMDD
	Store_day_seq_no	number(20)		Unique store/day identifier
	Day	number(3)		Day
	Tran_no_generated	char(6)		If NULL then blank

Record Name	Field Name	Field Type	Default Value	Description
	System_code	char(6)		System code
	Currency_rtl_dec	number(1)		Currency retail decimal places
Code Data	Code_type	char(4)		Unique code type identifier
	Code	char(6)		Unique code identifier
	Code_seq	number(4)		Unique code sequence identifier
Error Data	Error_code	char(25)		Error identifier
	Error_desc	char(255)		Error description
	Rec_solution	char(255)		Recommended solution (If NULL then 'there is no solution')
Credit Card Validation Data	Length	number(2)		Card number length
	From_prefix	number(6)		Start value for range of valid prefixes.
	To_prefix	number(6)		End value for range of valid prefixes.
	Tender_type_id	number(6)		Credit card ID
	Val_type	char(6)		Validation type. If NULL, than use "NONE".
Store POS Data	Store	number(10)		Store identifier
	Pos_type	char(6)		POS type identifier
	Start_tran_no	number(10)		First transaction number produced. Right justified and zero padded.
	End_tran_no	number(10)		Last transaction number produced. Right justified and zero padded.
Tender Type Data	Tender_type_group	char(6)		Tender type group
	Tender_type_id	number(6)		Tender type identifier. Right justified and zero padded.
	Tender_type_desc	char(40)		Tender type description.
Merchant Code Data	Non_merch_code	char(6)		Code identifying a non-merchandise cost that can be added to an invoice.

Record Name	Field Name	Field Type	Default Value	Description
Partner Data	Partner_type	char(6)		Specifies the type of partner. Valid values are Bank 'BK', Agent 'AG', Freight Forwarder 'FF', Importer 'IM', Broker 'BR', Factory 'FA', Applicant 'AP', Consolidator 'CO', and Consignee 'CN', Supplier hierarchy level 1 'S1', Supplier hierarchy level 2 'S2', Supplier hierarchy level 3 'S3'.
	Partner_id	char(10)		Partner vendor number.
Supplier Data	Supplier	number(10)		Supplier vendor number.
	Sup_status	char(1)		Determines whether the supplier is currently active. Valid values include: 'A' for an active supplier or 'I' for an inactive supplier.
Employee Data	Store	number(10)		Store number.
	Pos_id	char(10)		The POS ID of the employee.
	Emp_id	char(10)		The employee ID of the employee.
Promotion Data	Promotion	number(11)		Promotion number

Scheduling Considerations

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

Restart Recovery

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

Sales Audit Voucher Upload (savouch)

Functional Area

Automated Audit

Module Affected

savouch.pc

Design Overview

Gift certificates are unique in that they are sold by the retailer and then redeemed by the consumer. They are both items and tender. Gift certificate tracking is an important component of loss prevention and profit tracking. Gift certificate validation ensures that gift certificates are valid (and not counterfeited or previously redeemed) and that merchandise leaving the store has been paid for. Gift certificate tracking allows retailers to see their gift certificate liability, or how much in dollar terms they have outstanding and how much merchandise they owe consumers.

As gift certificates can enter the Sales Audit system as either items or tender, processing must be done to match up the sales and redemptions. This program will be used to aggregate gift certificate and voucher records.

Some retailers assign gift certificates to a given store, meaning that before a gift certificate is sold at a store, it is assigned to a given store. When a retailer assigns a gift certificate to a given store, a record is written to the database. When the gift certificate is then sold by the store and redeemed by the consumer, this existing record must be updated to include the sale and redemption information. Some retailers choose not to assign gift certificates and instead simply sell gift certificates. In this case, the record will be inserted into the database when the gift certificate is sold and updated when the gift certificate is redeemed.

This program will compare records in the files (produced by either saimptlog or a gift certificate application) to the database. If a record for the voucher does not exist on the database, the record should be inserted. If the voucher already exists on the database, the record should be updated with the appropriate information.

This program will use arrays to insert into and update the database.

Scheduling Constraints

Pre/Post Logic Description

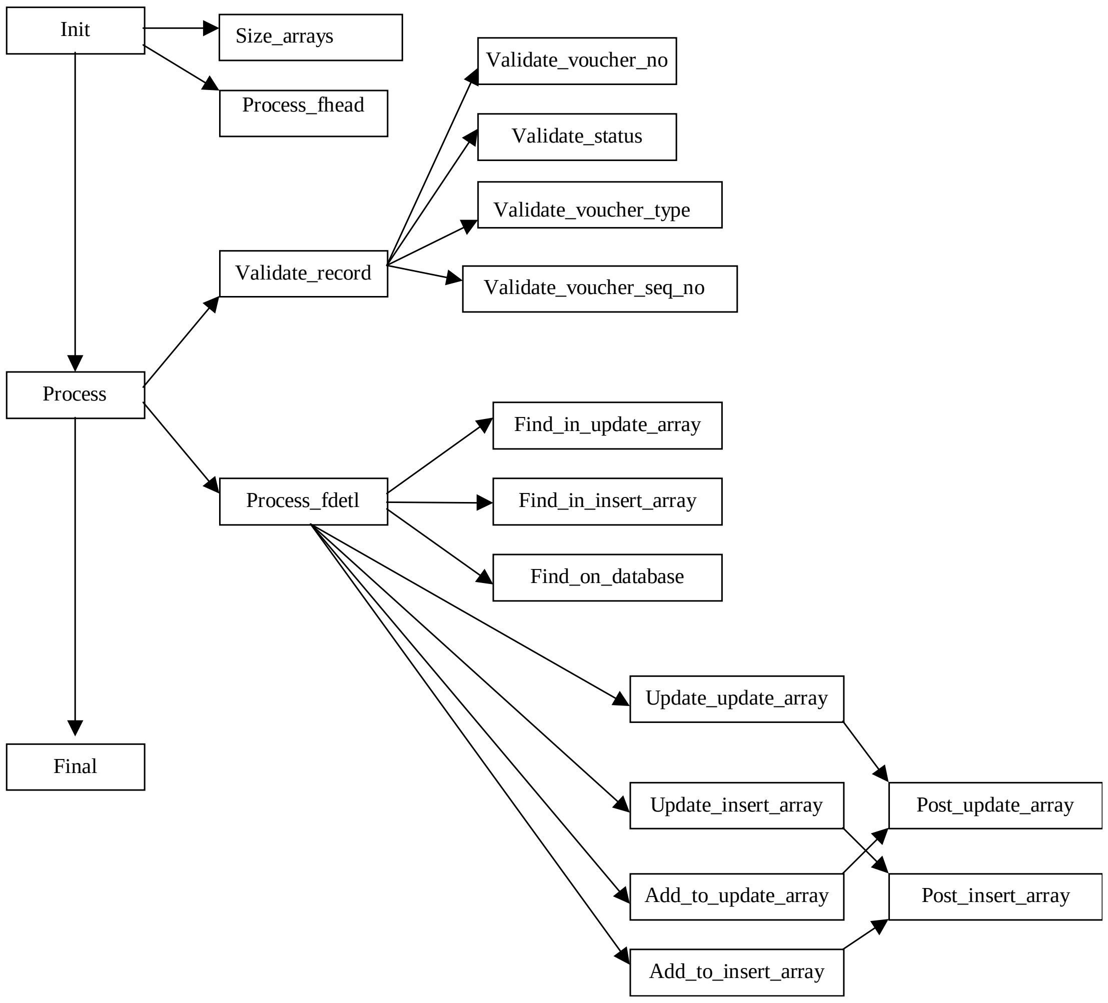
Processing Cycle	This program should be run daily, after saimptlog.
Scheduling Diagram	
Pre-Processing:	saimptlog.pc
Post-Processing	
Threading Schema	As this program processes an input file, it will have only one thread.

Restart Recovery

Logical Unit of Work (Recommended Commit Checkpoints) *Driving Cursor*

The logical unit of work for the voucher upload module is the voucher detail record (FDETL).

Program Flow/Technical Overview



Structure Chart

Shared Modules

Listing of all externally referenced functions and stored procedures and description of usage

Function Level Description

All Database Interactions Required and Error Handling Considerations

init()

This function initializes the necessary restart recovery variables, set up a reject file and open the input file, and call process_fhead.

process_fhead()

This function reads the first record - FHEAD - and validates that it contains the appropriate file type definition.

size_arrays()

This function allocates memories for the insert and update arrays used in process.

process()

This function contains the main process loop, which retrieves records from the file, and validates them by calling the function validate_record(). If the validation succeeds, the record is processed by process_fdetl(). If the validation returns a non-fatal error write to the reject file, it continues to process record. Two arrays are needed – one to update records that already exists on the database and another to insert new records into the database. The arrays are initialized by size_arrays, which should be called at the beginning of the function. This function should also insure that all records are posted to the database, so after all records have been processed, it should call post_insert_array() and post_update_array().

validate_record()

This function accepts a record as input and calls the validation sub-functions: validate_voucher_no(), validate_status(), validate_voucher_type(), and validate_voucher_seq_no. If any of these functions return a non-fatal error, this function should also return a non-fatal error.

validate_voucher_no()

This function accepts the voucher_no field from the input files and ensures that the voucher_no from the input file is either all blank or all numeric. If voucher_no is neither of these, the function should return a non-fatal error.

validate_status()

This function takes the status field from the input file and checks that the status field contains a SAVS_A, SAVS_I, or SAVS_R for assigned, issued, or redeemed. If the status is not one of these values, the function should return a non-fatal error. The function should then ensure that if the status is SAVS_A, that either ass_date or ass_store are either all blank or all numeric. If the status is SAVS_I, the function should ensure that the 'iss_' values are either all blank or all numeric. If the status is SAVS_R, the function should ensure that the 'red_' values are either all blank or all numeric.

validate_type()

This function takes the type field from the input file and check that it contains either the tender type id of 4000 or 4030 (gift certificate or voucher). If the type is not 4000 or 4030, this function should return a non-fatal error.

validate_voucher_head_no

This function takes the voucher_seq_no from the input file and checks that voucher_seq_no is not NULL. If the voucher_seq_no is NULL, the function should return a non-fatal error.

process_fdetl()

This function processes an inputted FDETL record. It determines whether the record needs to be inserted into the database or used to update an existing database record. To determine whether the record should be inserted or updated, several functions must be called.

If a record for the current voucher is already in the update array, additional information needs to be added to the record, rather than creating a new record. Check to see if information for the voucher already exists on the update array by calling find_in_update(). If it finds a record in the update array, update_update_array() should be call to add the new data to the correct update record.

If a record for the voucher does not exist on the update array, check the insert array by calling find_in_insert(). If the record already exists on the insert array, the additional information can be added to the record by calling update_insert_array().

If the voucher is not in either of the arrays, check to see if the voucher already exists on the database by calling find_on_database(). If it exists on the database, the record retrieved from the database should be added to the update array by calling add_to_update_array(). Then calling update_update_array should aggregate the information from the file(). If a record for the voucher does not exist on the database, add_to_insert_array() should be called to add the record from the file to the insert array.

find_in_update_array()

This function takes a voucher_no as input and searches the update array for it. If the voucher is in the array, the function should return the row/index number (the location of the record in the array). If the voucher is not in the array, it should return –1.

update_update_array()

This function updates the update array (hence the name). It takes the record from the file and the array index of the record already on the array for the voucher. It copies any values that are not null from the record into null values on the array. The status fields require additional logic – the 'greater' of the two statuses should end up on the database ('I' trumps 'A', 'R' trumps all).

find_in_insert_array()

This function takes a voucher_no as input and searches the insert array for it. If the voucher is in the array, the function should return the row/index number (the location of the record in the array). If the voucher is not in the array, it should return –1.

update_insert_array()

This function updates the insert array (hence the name). It takes the record from the file and the array index of the record already on the array for the voucher. It will copy any values that are not null from the record into null values on the array. The status fields require additional logic – the 'greater' of the two statuses should end up on the database ('I' trumps 'A', 'R' trumps all).

find_on_database()

This function queries the database for all information from the sa_voucher table for a given voucher_no. This function should return 1 if it finds a record on the database and 0 if it does not.

add_to_update_array()

This function adds a record to the update array. This record consists of the information fetched from the database. If the array becomes too large, call post_update_array to post the information to the database and clear out the array. Make sure that fields that have no values (are NULL) are blanked out with an empty string.

add_to_insert_array()

This function adds a record to the insert array. This record consists of the information from the file. If the array becomes too large, call post_insert_array to post the information to the database and clear out the array. Make sure that fields that have no values (are NULL) are blanked out with an empty string.

post_update_array()

The function posts the updated records to the sa_voucher table using the information from the update array. It then clears out the array setting the array counter back to zero.

post_insert_array()

The function posts the inserted records to the sa_voucher table using the information from the insert array. It then clears out the array setting the array counter back to zero.

final()

This function calls restart_close and closes the input and rejects files.

I/O Specification

All File Layouts Input and Output

The input to this module is described in the Batch Design for the saimptlog.pc program. The output of this program is written to the sa_vouch table.