RMS ETL10.1

Operations Guide

Retek®

WWW.RETEK.COM

HELPING THE RETAIL INDUSTRY CREATE, MANAGE AND FULFILL CONSUMER DEMAND™

Corporate Headquarters:

Retek Inc.
Retek on the Mall
950 Nicollet Mall
Minneapolis, MN 55403

888.61.RETEK (toll free US)
+1 612 587 5000

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

Sales Enquiries:
+44 (0)20 7563 46 46
Fax: +44 (0)20 7563 46 10

Customer Support

Customer Support hours:

Customer Support is available 7x24x365 via e-mail, phone, and Web access.

Depending on the Support option chosen by a particular client (Standard, Plus, or Premium), the times that certain services are delivered may be restricted. Severity 1 (Critical) issues are addressed on a 7x24 basis and receive continuous attention until resolved, for all clients on active maintenance.

Contact Method	**Contact Information**
Internet (ROCS)	www.retek.com/support Retek's secure client Web site to update and view issues
E-mail	support@retek.com
Phone	US & Canada: 1-800-61-RETEK (1-800-617-3835) World: +1 612-587-5800 EMEA: 011 44 1223 703 444 Asia Pacific: 61 425 792 927
Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

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.

Contents

Chapter 1 – Introduction

The Retek Merchandising System (RMS) is the retail client's central merchandising transactional processing system. RMS is the principle source of the foundation or dimensional data needed in some of the Retek 10 suite of products.

RMS supplied dimensions for these extracts include the following: Company, Organizational hierarchy, Merchandise hierarchy (including Product), Calendar hierarchy, and Weekly and Daily Sales information.

RMS supplied loads include the following: daily forecast demand and weekly forecast demand.

A program overview of standard programming features is described as well as the program reference lists of these shared extracts. The application-programming interface (API) describes the flat file specifications in Appendix A.

Technical architecture

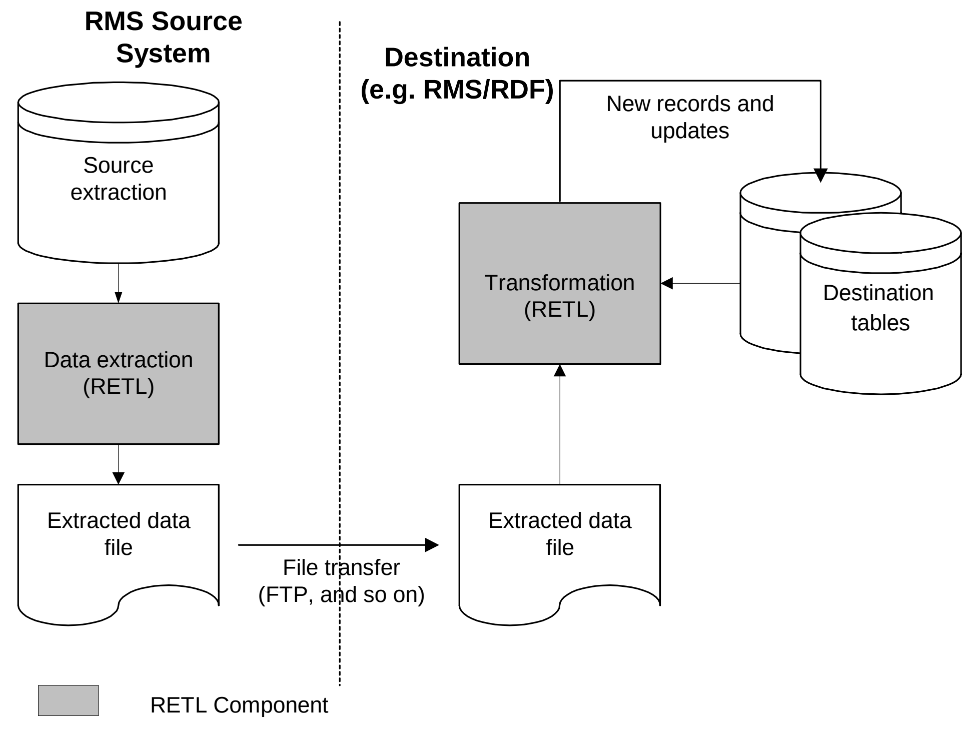
The modules work in conjunction with the Retek Extract Transform and Load (RETL) framework. This architecture optimizes a high performance data processing tool that allows database batch processes to take advantage of parallel processing capabilities.

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

Extract Transform and Load (ETL) processing

The following diagram illustrates the extraction processing architecture. Instead of managing the change captures as they occur in the source system during the day, the process involves extracting the current data from the source system. The extracted data is output to flat files. These flat files are then available for consumption by products such as Retek Data Warehouse (RDW) and Retek Demand Forecasting (RDF).

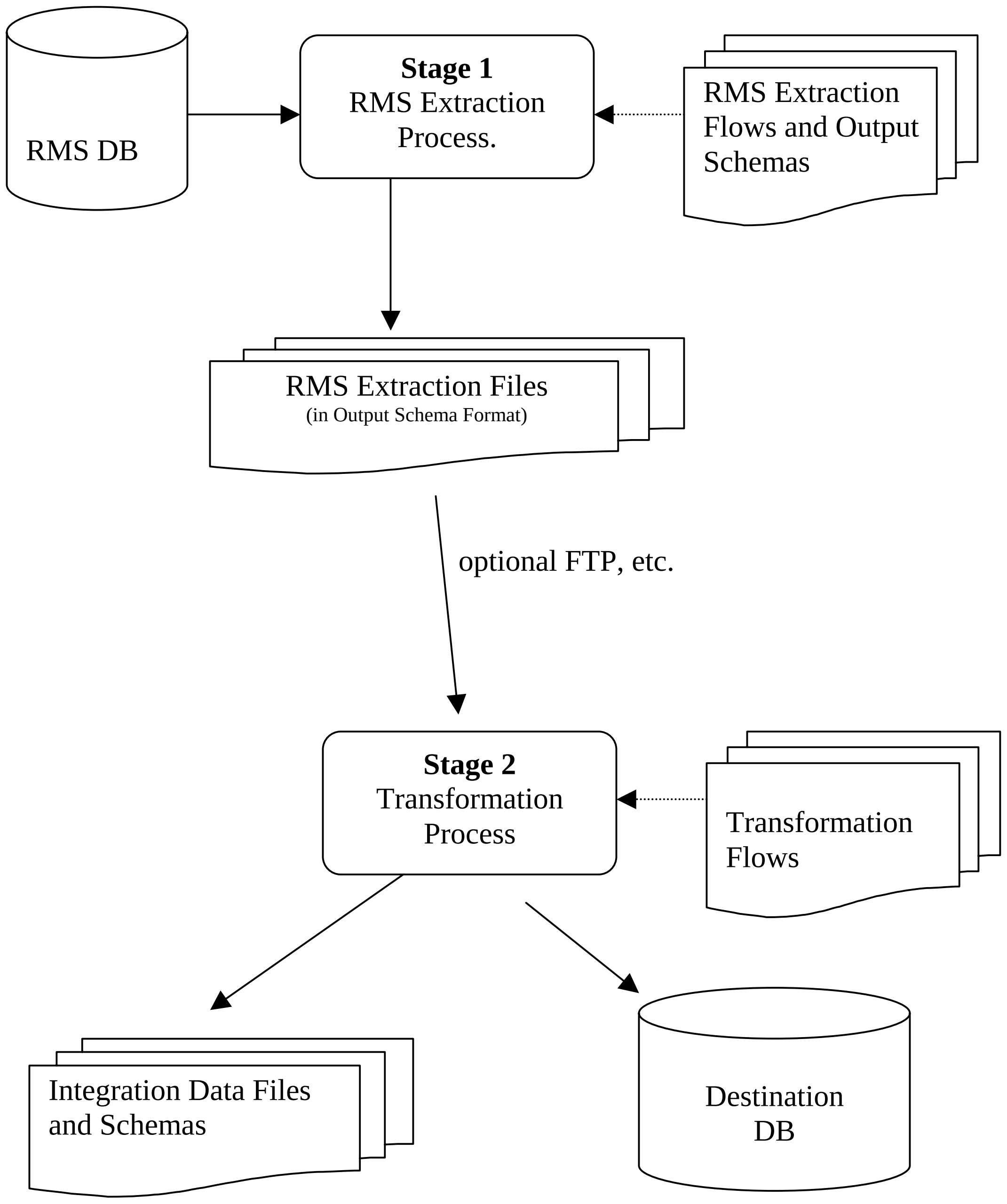
The target system, (RDW or RDF, for example), has its own way of completing the transformations and loading the necessary data into its system, where it can be used for further processing in the environment.



Data Movement Architecture

Design

The components for this solution will adhere to the following general architecture:



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

Stage 2 introduces a flow specific to the destination. In this case flows for the RDF/RPAS product designed to transform the data such that RDF can import the data properly.

Chapter 2 – Program overview

This chapter summarizes the RETL program features utilized in the RMS Extractions (RMSE). More information about the RETL tool is available in the latest RETL Programmer's Guide.

Installation

Select a directory where you would like to install RMS ETL. This directory (also called MMHOME) is where all of the RMS ETL files will be extracted.

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

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

Configuration

RETL

Before trying to configure and run RMS ETL, install RETL version 10.3 or later, which is required to run RMS ETL. Run the "verify_retl" script (included as part of the RETL installation) to ensure that RETL is working properly before proceeding.

RETL User and Permissions

RMS ETL should be installed and run as the RETL user. Additionally, the permissions should be setup as per the RETL Programmer's Guide. RMS ETL will read data, create, delete and update tables. (This is done for example to ensure that weekly sales data is not pulled multiple times on subsequent extractions.) If these permissions are not setup properly – extractions will fail.

Environment Variables

In addition to the RETL environment variables see the Programmer's Guide for your version of RETL. You will need to set MMHOME to your base directory for RMS ETL. This is the top level directory that you selected during the installation process (see Installation above). So in your .kshrc you should add a line like the following:

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

rmse_config.env

There are a couple variables you will need to change depending upon your local settings:

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

Also, you will need to set the environment variable PASSWORD in either the rmse_config.env, .kshrc or some other location that can be included via one of those two means. For example, adding this line to the rmse_config.env will cause the password "bogus" to be used to log into the database:

```
export PASSWORD=retek
```

Program features

The extraction programs are written in the RETL framework and include the following features:

- Program return code
- Program status control files
- Restart and recovery
- Message logging
- Program error file
- Reject files
- Schema files
- Command line parameters

Program return code

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

Program status control files

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

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

File Naming conventions

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

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

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

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

```
$MMHOME/error/invildex.invilddm.txt.status.20010105
```

Restart and recovery

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

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

RMS ETL restart and recovery

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

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

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

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

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

Bookmark file

The name and directory of the restart and recovery bookmark file is set in the configuration file (rmse_config.env). The directory defaults to $MMHOME/rfx/bookmark. The naming convention for the bookmark file itself defaults to the following “dot” separate file name:

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

For example, the bookmark flag for the `invildex` program would be written to the following file for the batch run of January 5, 2001:

```
$MMHOME/rfx/bookmark/invildex.invilddm.txt.bkm.20010105
```

Message logging

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

Daily log file

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

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

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

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

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

Format

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

```
cusdemogdm 13:20:01: Program Starting...

cusdemogdm 13:20:05: Build update and insert data.

cusdemogdm 13:20:13: Analyze table
rdw10dev.cust_demog_dm_upd

cusdemogdm 13:20:14: Insert/Update target table.

cusdemogdm 13:20:23: Analyze table rdw10dm.cust_demog_dm

cusdemogdm 13:20:27: Program Completed...
```

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

Program error file

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

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

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

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

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

```
$MMHOME/error/slsildmdm.slsildmdm.txt.20010105
```

RMSE reject files

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

Note: See documentation on individual flows to determine whether reject files will cause the flow as a whole to fail. Sometimes rejections are considered failures sometimes they are considered normal from an operational standpoint.

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

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

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

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

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

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

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

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

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

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

Schema files

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

Command line parameters

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

RMSE

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

See “Program reference lists” for a detailed listing of all programs and their command line parameters.

Typical run and debugging situations

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

Example

To run rmse_stores.ksh:

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

```
%rmse_stores.ksh
```

If the module runs successfully, the following results:

1 **Log file:** Today's log file, 20010309.log, contains the messages "Program started …" and "Program completed successfully" for rmse_stores.

2 **Data:** The rmse_stores.dat file exists in the data directory and contains the extracted records.

3 **Schema:** The rmse_stores.schema file exists in the schema directory and contains the definition of the data file in #2 above.

4 **Error file:** The program's error file, rmse_stores.20010309, contains the standard RETL flow (ending with "All threads complete" and "Flow ran successfully") and no additional error messages.

5 **Program status control:** The program status control file, rmse_stores.status.20010309, does not exist.

6 **Reject file:** The reject file, rmse_stores.rej.20010309, does not exist.

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

1 **Log file:** Today's log file, 20010309.log, does not contain the "Program completed successfully" message for rmse_stores.

2 **Data:** The rmse_stores.dat file may exist in the data directory but may not contain all the extracted records.

3 **Schema:** The rmse_stores.schema file exists in the schema directory and contains the definition of the data file in #2 above.

4 **Error file:** The program's error file, rmse_stores.20010309, may contain an error message.

5 **Program status control:** The program status control file, rmse_stores.status.20010309, exists.

6 **Reject file:** The reject file, rmse_stores.status.20010309, does not exist because this module does not reject records.

7 **Bookmark file:** The bookmark file, rmse_stores.bkm.20010309, does not exist because this module does not utilize restart and recovery.

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

1 Determine and fix the problem causing the error.

2 Remove the program’s status control file.

3 Change directories to $MMHOME/rfx/src. At a Unix prompt, enter:

```
%rmse_stores.ksh
```

Chapter 3 – Batch processing

The following explains the order constraints of the batch schedule. This section includes:

- The overall batch schedule;

The module inter-dependencies between the RMS batch and the RMSE batch.

Setting up the batch schedule

Note: The number of modules that can be run in parallel at any given time is dependent upon the client's hardware capacity.

On a typical batch production run, the pre-batch maintenance modules must always run first. What runs next, as long as the client follows the batch dependencies in the flow diagrams, is up to the client.

In short, "pre_rmse.ksh" must be run prior to running other scripts (see detailed notes below). All other extract modules can be run in parallel. (Note, however, that some RMSE modules have RMS pre-dependencies, and some RMS modules are dependent upon RMSE modules.) In other words, product extract modules can be run in parallel with organization extract modules. Because of the extract modules' ability to be run in parallel, it does not matter which module goes first (assuming no interdependencies are noted on the batch flows below).

The batch flows on the following pages are best read from top to bottom. Such a review of the RMSE batch schedule allows clients to both setup module dependencies and to optimize their batch window through the concurrent running of unrelated modules.

rmse_config.env settings

On the RMSE side, make sure to review the environmental parameters in the rmse_config.env file before executing batch modules.

Steps to configure RETL

1. Log in to the Unix server with a Unix account that will run the RETL scripts. For instance, the `rmsedev` account is used in RDW.
2. Change directories to `<base_directory>/rmse10.1/dev/rfx/etc`.
3. Modify the rmse_config.env script:
   a. Change the DBNAME variable to the name of the RMS database.
   b. Change the RMS_OWNER variable to the username of the RMS schema owner.
   c. Change the BA_OWNER variable to the username of the RMSE batch user.

RMS and Destination batch schedule

The (RMSE) extraction modules run in the RMS batch cycle and are dependent on some RMS modules to provide data for processing (see the descriptions of the individual modules for details). Some RMS modules are dependent on RMSE modules. Most RMSE extraction programs run after Phase 2 of the RMS batch cycle is completed. All RMSE modules must run before the RMS vdate is incremented.

Refer to the destination (e.g. RDW/RDF) transformation and load modules for the product of interest to see how batch jobs should be setup to work appropriately with RMSE.

Chapter 4 – Program reference lists

This chapter serves as a reference to the following RMSE programs and reference information:

- Extraction (RETL Korn shell scripts)
- Maintenance (RETL Korn shell scripts)

By reviewing Chapter 3, "Program flow diagrams", along with this chapter and Appendix A, "API flat file specifications", the client should be able to track, down to the table and column level, all the extraction data that flows out of RMS.

Extraction programs

Extraction Name	Tables Extracted	Fields Extracted	Target File or Table	Target Field	Field Type	Field Length	Notes
rmse_store.ksh	store	store	rmse_store.dat	store	Number(10)	11	
		store_name		store_name	Varchar2(20)	20	
		district		district	Number(4)	5	
		store_close_date		store_close_date	Date	8	
		store_open_date		store_open_date	Date	8	
		store_class		store_class	Varchar2(1)	1	
	code_detail	store_class_description		store_class_description	Varchar2(40)	40	joined with store.store_class, code type 'CSTR'
		store_format		**store_format**	Number(4)	5	
	store_format	format_name		format_name	Varchar2(20)	20	joined with store.store_format
rmse_wh.ksh	wh	wh	rmse_wh.dat	wh	Number(10)	11	
		wh_name		wh_name	Varchar2(20)	20	
		forecast_wh_ind		forecast_wh_ind	Varchar2(1)	1	
		stockholding_ind		stockholding_ind	Varchar2(1)	1	

Extraction Name	Tables Extracted	Fields Extracted	Target File or Table	Target Field	Field Type	Field Length	Notes
Rmse_orghier.ksh	district	district	rmse_orghier.dat	district	Number(4)	5	
		district_name		district_name	Varchar2(20)	20	
		region		**region**	Number(4)	5	
	region	region_name		region_name	Varchar2(20)	20	joined with district.region
		area		**area**	Number(4)	5	
	area	area_name		area_name	Varchar2(20)	20	joined with region.area
		chain		**chain**	Number(4)	5	
	chain	chain_name		chain_name	Varchar2(20)	20	joined with area.chain
	comphead	company		company	Number(4)	5	merged (should be a single row)
		co_name		co_name	Varchar2(20)	20	
Rmse_merchhier.ksh	subclass	subclass	rmse_merchhier.dat	subclass	Number(4)	5	
		sub_name		sub_name	Varchar2(20)	20	
	class	class		class	Number(4)	5	joined with subclass.class
		class_name		class_name	Varchar2(20)	20	
	deps	dept		dept	Number(4)	5	joined with class.dept
		dept_name		dept_name	Varchar2(20)	20	
	groups	group_no		group_no	Number(4)	5	joined with dept.group_no
		group_name		group_name	Varchar2(20)	20	
	division	division		division	Number(4)	5	joined with groups.division
		div_name		div_name	Varchar2(20)	20	
Rmse_sups.ksh	sups	supplier	rmse_supplier.dat	supplier	Number(10)	11	
		sup_name		sup_name	Varchar2(32)	32	
Rmse_domain.ksh	domain	domain	rmse_domain.dat	domain	Number(2)	3	
	domain_dept	dept		dept	Number(4)	5	
	domain_class	class		class	Number(4)	5	Also domain_class.dept
	domain_subclass	subclass		subclass	Number(4)	5	Also domain_subclass.dept and Domain_subclass.class
	Domain_dept\ class\ subclass	load_sales_ind		load_sales_ind	Varchar2(1)	1	

Extraction Name	Tables Extracted	Fields Extracted	Target File or Table	Target Field	Field Type	Field Length	Notes
rmse_item_master.ksh	item_master	item	rmse_item_master.dat	item	Varchar2(25)	25	This is the item master extrac
		item_desc		item_desc	Varchar2(100)	100	
		item_parent		item_parent	Varchar2(25)	25	
		item_grandparent		item_grandparent	Varchar2(25)	25	
		subclass		subclass	Number(4)	5	
		class		class	Number(4)	5	
		dept		dept	Number(4)	5	
		forecast_ind		forecast_ind	Varchar2(1)	1	
	item_supplier	supplier		supplier	Number(10)	11	
rmse_item_loc_hist.ksh	item_loc_hist	item	rmse_item_loc_hist.dat	item	Varchar2(25)	25	
		loc		loc	Number(10)	11	
		eow_date		eow_date	Date	8	
		sales_issues		sales_issues	Number(12, 4)	14	
rmse_tran_data_history.ksh	tran_data_history	item	rmse_tran_data_history.dat	item	Varchar2(25)	25	
		store		store	Number(10)	11	
		wh		wh	Number(10)	11	
		tran_date		tran_date	Date	8	
		units		units	Number(12, 4)	14	
rmse_item_loc_soh.ksh	item_loc_soh	item	rmse_item_loc_soh.dat	item	Varchar2(25)	25	
		loc		Loc	Number(10)	11	
		stock_on_hand		stock_on_hand	Number(12, 4)	14	
	Period	vdate		*extract_date*	Date	8	Vdate global variable - date of extraction

Extraction Name	Tables Extracted	Fields Extracted	Target File or Table	Target Field	Field Type\Length	Field Length	Notes
rmse_weekly_sales.ksh	item_master	item	rmse_weekly_sales.dat	item	Varchar2(25)	25	This is the item master extract
	item_loc_soh	loc		loc	Number(10)	11	
	item_loc_hist	eow_date		eow_date	Date	8	
		sales_issues		sales_issues	Number(12, 4)	14	
		sales_type		sales_type	Varchar2(1)	1	
	item_loc_soh	rowid		row_id	Varchar2(18)	18	
	domain_subclass domain_class domain_dept	domain_id*		domain_id	Number(2)	3	Table will depend on domain le Class, Subclass)
rmse_daily_sales.ksh	tran_data_history/ if_tran_data	loc	rmse_daily_sales.dat	Loc	Number(10)	11	This is the item master extract
	item_loc_soh/ if_tran_data	item		Item	Varchar2(25)	25	
	tran_data_history/ if_tran_data	tran_date		tran_date	Date	8	
		sum(units)		sum_units	Number(12, 4)	14	
		sales_type		sales_type	Varchar2(1)	1	
		tran_code		tran_code	Number(2)	3	
	domain_subclass domain_class domain_dept	domain_id*		domain_id	Number(2)	3	Table will depend on domain le Class, Subclass)
rmse_stock_on_hand.ksh	item_loc_soh	item	rmse_stock_on_hand.dat	item	Varchar2(25)	25	
		loc		loc	Number(10)	11	
		stock_on_hand		stock_on_hand	Number(12,4)	14	

Ftmednld.pc - Calendar Hierarchy Download

The Calendar Hierarchy download will continue to be a Pro*C batch extract. Please reference the detailed design for further information. The file produced by this extract will need to be transferred to a location where the RDF Transform scripts can access it.

Load programs

Weekly Forecasted Demand

RDF Batch Name:	export_fcst.sh
File Location:	from_rpas
File Names:	wfdemand.##
Example:	wfdemand .01

Field Name	Start Position	Width	Format	RMS Table	Load	Schema Field
End-of-week Date	1	8 char	yyyymmdd	Item_forecast.eow_date	weekly_forecast	eow_date
Item ID	9	20 char	Alpha	Item_forecast.item	weekly_forecast	item
Store/Warehouse ID	29	20 char	Alpha	Item_forecast.loc	weekly_forecast	location
Quantity	49	12 char	Numeric	Item_forecast.forecast_sales	weekly_forecast	fcst_sales
Standard Deviation	61	12 char	Numeric	Item_forecast.forecast_std_dev	weekly_forecast	forecast_std_dev

- The numeric fields are zero-padded and the decimal point is omitted, but the quantities have a 4-digit decimal part.

Example:

20021119000000000000123456780000000000000000001234000000121234000000345678

This indicates:

Date:	19 November 2002
Item:	12345678
Store:	1234
Quantity:	12.1234
Std. Dev.:	34.5678

- The format of the export can be modified through the RDF client in the Forecast Export Administration workbook – which means that we can modify the format of the file for easier import.
- The Item and Store/Warehouse fields are left justified.

Daily Forecasted Demand

RDF Batch Name:	export_fcst.sh
File Location:	from_rpas
File Names:	dfdemand.##
Example:	dfdemand.01

Field Name	Start Position	Width	Format	RMS Tables	Load	Schema.Field
Date	1	8 char	yyyymmdd	Daily_item_forecast.data_date	weekly_forecast	data_date
Item ID	9	20 char	Alpha	Daily_item_forecast.item	weekly_forecast	item
Store/Warehouse ID	29	20 char	Alpha	Daily_item_forecast.loc	weekly_forecast	location
Quantity	49	12 char	Numeric	Daily_item_forecast.forecast_sales	weekly_forecast	forecast_sales
Standard Deviation	61	12 char	Numeric	Daily_item_forecast.forecast_std_dev	weekly_forecast	forecast_std_dev

- The numeric fields are zero-padded and the decimal point is omitted, but the quantities have a 4-digit decimal part.

Example:

200211190000000000000123456780000000000000000012340000000121234000000345678

This indicates:

Date:	19 November 2002
Item:	12345678
Store:	1234
Quantity:	12.1234
Std. Dev.:	34.5678

- The format of the export can be modified through the RDF client in the Forecast Export Administration workbook – which means that we can modify the format of the file for easier import.
- The Item and Store/Warehouse fields are left justified.

Load Name	Fields Extracted	Target File or Table	Tables Loaded	Target Field	Field Length	Notes
rmsl_forecast.ksh	item	rmsl_item_forecast.dat	itemforecast (daily_item_forecast)	item	Varchar2(25)	This is the first load.
	loc			loc	Number(10)	
	eow_date (data_date)			eow_date	Date	
	forecast_sales			forecast_sales	Number(12, 4)	
	forecast_std_dev			forecast_std_dev	Number(12, 4)	

rmsl_forecast.ksh

This script can be run for either weekly or daily forecasting.

Maintenance programs

Program	Functional Area	Module Type	External Data Source	Source Table or File	Schema File	Target File or Table	Arguments	Notes
pre_rmse.ksh	Pre-RMS Extraction Maintenance	Maintenance	RMS	PERIOD, SYSTEM_OPTIONS, SYSTEM_VARIABLES, CURRENCY_RATES		class_level_vat_ind.txt, consolidation_code.txt, domain_level.txt, last_eom_date.txt, max_backpost_days.txt, multi_currency_ind.txt, prime_currency_code.txt, prime_exchng_rate.txt, stkldgr_vat_incl_retl_ind.txt, vat_ind.txt, vdate.txt		This module expects these text files to exist in $MMHOME/rfx/etc when it runs. Text files containing default values for the very first run are included in the installation process.

Appendix A – Application programming interface (API) flat file specifications

This appendix contains APIs that describe the file format specifications for all text files that serve as the interface between source systems and target systems. For example, these APIs control the formatting of the following:

- The dimension data extracted by the DWI code

In addition to providing individual field description and formatting information, the APIs provide basic business rules for the incoming data.

API format

Each API contains a business rules section and a file layout. Some general business rules and standards are common to all APIs. The business rules are used to ensure the integrity of the information held within the target system. In addition, each API contains a list of rules that are specific to that particular API.

File layout

- Field Name: Provides the name of the field in the text file.
- Description: Provides a brief explanation of the information held in the field.
- Max Column Length: Identifies the maximum length possible for a field. A field may not exceed this length.
- Data Type/Format: Data type identifies one of three valid data types: character, number, or date:
  - Character: Can hold letters (a,b,c…), numbers (1,2,3…), and special characters ($,#,&…)
  - Numbers: Can hold only numbers (1,2,3…)
  - Date: Holds a specific year, month, day combination

Any required formatting for a field is conveyed in the Format section. For example, Number(18,4) refers to number precision and scale. The first value is the precision and always matches the maximum column length; the second value is the scale and specifies how many digits exist to the right of the decimal point.

- Required Field: Identifies whether the field can hold a null value. This section holds either a ‘yes’ or a ‘no’. A ‘yes’ signifies the field may not hold a null value. A ‘no’ signifies the field may, but is not required to, hold a null value.

General business rules and standards common to all APIs

- Complete ‘snapshot’ of dimension data:
  A majority of RDW’s dimension code requires a complete view of all current dimensional data (irregardless of whether the dimension information has changed) in order to successfully capture the correct data on the target table. If a complete view of the dimensional data is not provided in the text file, invalid or incorrect dimensional data can result. For instance, not including an active item in the prditmdm.txt file causes that item to be closed (as of the extract date) in the data warehouse. When a sale for the item is processed, the fact program will not find a matching ‘active’ dimension record. Therefore, it is essential, unless otherwise noted in each API’s specific business rules section, that a complete snapshot of the dimensional data be provided in each text file.

- Leading/trailing values:
  Values entered into the text files are the exact values processed and loaded into the datamart tables. Therefore, the values with leading and/or trailing zeros, characters, or nulls are processed as such. RDW does not strip any of these leading or trailing values, unless otherwise noted in the individual API’s business rules section.

- Delimiters:

  Note: Make sure the delimiter is never part of your data.

  - Within dimension text files, each field must be separated by a pipe (|) character, for example a record from prddivdm.txt may look like the following:

    ```
    1000|1|Homewares|2006|Henry Stubbs|2302|Craig Swanson
    ```

  - Within facts text files, each field must be separated by a semi-colon character (;). For example a record from exchngratedm.txt may look like the following:

    ```
    WIS;20010311;1.73527820592648544918
    ```

    See the latest RETL Programmer’s Guide for additional information.

- End of Record Carriage Return:
  Each record in the text file must be separated by an end of line carriage return. For example, the three records below, in which each record holds four values, should be entered as:

  ```
  1|2|3|4
  5|6|7|8
  9|10|11|12
  ```

  and not as a continuous string of data, such as:

  ```
  1|2|3|4|5|6|7|8|9|10|11|12
  ```

- Character format:
  All API’s should contain ASCII text characters only.