



Retek

Retek Merchandising System Operations Guide

Retek Merchandising System™

The software described in this documentation is furnished under a license agreement and may be used only in accordance with the terms of the agreement.

Copyright Notice

Copyright © 2001 by Retek Inc.

All rights reserved.

No part of this documentation may be reproduced or transmitted in any form or by any means without the express written permission of Retek Inc., 801 Nicollet Mall, Suite 1100, Minneapolis, MN 55402.

Information in this documentation is subject to change without notice.

Trademarks

Retek Merchandising System is a trademark of Retek Inc.

All other product names mentioned are trademarks or registered trademarks of their respective owners and should be treated as such.

Policy on Retek End User Documentation

As a standard policy, Retek provides read-only copies of our documentation to customers and other third parties. Unauthorized changes to the documentation may not accurately describe how our software functions. Retek Customer Care is unable to support any changed end user documentation that was not authorized or approved by Retek.

Printed in the United States of America.

Customer Support

Customer Support hours:

8 AM to 5 PM Central Standard Time (GMT-6), Monday through Friday,
excluding Retek company holidays (in 2001: Jan. 1, May 28, July 4, Sept. 3,
Nov. 22, Nov. 23, Dec. 24, and Dec. 25).

Customer Support emergency hours:

24 hours a day, 7 days a week.

Contact Method	Contact Information
Phone	US & Canada: 1-800-61-RETEK (1-800-617-3835) World: + 1 612-630-5800
Fax	(+1) 612-630-5710
E-mail	support@retек.com
Internet	www.retek.com/support Retek's secure client Web site to update and view issues
Mail	Retek Customer Support Midwest Plaza 801 Nicollet Mall Suite 1100 Minneapolis, MN 55402

When contacting Customer Support:

- Always fill out an Issue Report Form before submitting issues to Retek (request forms from Customer Support if necessary).
- Provide a completely updated Customer Profile.
- Have a single resource per product responsible for coordination and screening of Retek issues.
- Respond to our requests for additional information in a timely manner.
- Use the Expert Web to submit and update your issues.
- Have a test system in place running base Retek code.

Contents

Chapter 1 - Introduction	1
What This Guide Shows You.....	1
What's in This Guide	2
What's Not in This Guide	3
Who This Guide is Written for.....	3
Where You Can Find More Information.....	3
How to Use This Guide.....	4
Other Good Things to Know.....	4
Chapter 2 - Glossary of Abbreviations, Acronyms, Terms.....	5
Chapter 3 - Interface Dataflow Overviews	9
RMS Batch Modules for Retek Trade Management.....	9
New in RTM 9.0	9
Batch Modules	10
Batch Modules at a Glance	12
Chapter 4 - Understanding EDI Batch Modules.....	13
Batch Modules at a Glance	13
Chapter 5 - The RMS and Retek Demand Forecasting 9.0 Interface	15
New to the RMS 9.0-RDF 9.0 Interface	15
Domains and their Purpose	16
Data Downloaded to RDF	16
Batch Modules That Download RMS's Organizational and Merchandise Hierarchies.....	16
Determine Data to Download to RDF.....	17
Upload and Process RDF Sales and Warehouse Issues Forecasts	18
Tables of Batch Modules	20

Chapter 6 - The RMS and Retek Predictive Planning 9.0 Interface 23

RMS Hierarchy Data Downloaded to RPP	24
RMS Weekly Sales Data Downloaded to RDF.....	24
Upload and Process RPP Open-to-Buy Data	25
Summary of Batch Modules	26

Chapter 7 - The RMS 9.0 and Retek Distribution Management 9.0 Interface 29

RMS Hierarchical (Foundation) Data Shared with RDM.....	29
Purchase Order, Allocation, and Receipt Data.....	29
Inventory Movement	29
New in RMS 9.0.....	30
RMS Batch Modules for Converting RDM Files	31
Summary of RMS Batch Modules in the RDM Interface.....	31

Chapter 8 - The RMS 9.0–RDW 9.0 Interface..... 37

Techniques Used in Data Transfer	37
Triggers for Dimension Data	37
Oracle SQL*Load Tool Used for Fact Data.....	37
Data in RDW and RMS: Defining Terms	38
RMS Batch Modules.....	40
RDW's Data Warehouse Interface Batch Modules.....	44
RMS Table Triggers and RDW Extraction Batch Modules:	48

Chapter 9 - Point of Sale Data Upload to RMS 51

The POS Upload Process	51
Preparing Transaction Data for the Upload	51
Uploading Transaction Data	51
Validating Total Amounts.....	53
The Validation Process	53
Posting Sales Transaction Data in RMS	54
A Special Note about Sales Audit 9.0 and POSUPLD.PC.....	55

Chapter 10 - Updating the POS System from RMS 57

The POS Download Process 57

Batch Modules in the POS Download Process 59

Chapter 11 - Retek Invoice Matching Batch Modules 61

Invoice Matching Process 62

Batch Modules at a Glance 63

Chapter 12 - The Retek Enterprise 9.0 - Oracle Financial Application 10.7 Interface Overview 65

A Note about Clearing Accounts 65

Knowing the Flow: Who “Owns” the Data..... 66

Currencies and Exchange Rates: from Oracle to RMS 67

Suppliers and Supplier Addresses: from Oracle to RMS 67

New for RMS 9.0: Support for Direct Store Delivery Expense Vendors as Partners..... 68

New for RMS 9.0: Selecting the Invoice Pay Site 68

Freight and Payment Terms: from Oracle to RMS 69

Purchase Orders: from RMS to Oracle 69

Receipts: from RMS to Oracle..... 69

Retek Invoice Matching (Invoice Matching) and Oracle Financials 70

Stock Ledger (sales transaction data): from RMS to Oracle..... 70

Retek Sales Audit Totals Export to Oracle General Ledger 71

Summary of Oracle Financials Application Batch Modules..... 72

Chapter 13 - Overview of Retek Sales Audit 9.0 Batch Processing 73

Store Day Defined..... 73

Preparation for the Data Import 73

Transaction Data Import and Validation..... 74

Total Calculations and Rules 76

Export Store Day Transaction Data to Applications.....	76
Transaction Data Exports and the Unit of Work.....	78
Retek Merchandise System (RMS) Export.....	78
Retek Data Warehouse (RDW) Export.....	78
Oracle Financial Application 10.7 General Ledger Export.....	79
Account Clearing House (ACH) Export.....	79
Universal Account Reconciliation System (UAR) Export.....	79
Retek Invoice Matching (RIM) Export.....	80
Escheatment Totals to RIM for Accounts Payable	80
Oracle Site Fuels Management (SFM) Export.....	80
Full Disclosure and Post-Export Changes.....	81
What Happens to Totals when Transactions Are Modified?	81
Adjustments Received from an Application	81
Chapter 14 - Retek Sales Audit Dataflow Diagrams	83
Summary of ReSA Batch Modules	85
ReSA 9.0 RTLOG Layout.....	88
Sales Audit Interface to Oracle 10.7'	117
'Metered Sales Volumes'	120
Meter Reading.....	121
Tank Dip	122
File layouts input and output.....	123
Chapter 15 - Batch Functional Overviews	127
Allocations	127
Functional Overview.....	127
Module Flow and Scheduling	127
Audit Trail.....	128
Functional Overview.....	128
Module Flow and Scheduling	128
Bill of Lading.....	129
Functional Overview.....	129
Module Flow and Scheduling	129
Calendar	130
Functional Overview.....	130
Module Flow and Scheduling	130

Clearance	131
Functional Overview	131
Module Flow and Scheduling	132
Color and Size Dimensions	133
Functional Overview	133
Module Flow and Scheduling	133
Complex Deals Maintenance	134
Functional Overview	134
Module Flow and Scheduling	134
Contracts	136
Functional Overview	136
Module Flow and Scheduling	137
Customs Entry (CE)	138
Functional Overview	138
Module Flow and Scheduling	138
EDI	139
Functional Overview	139
Module Flow and Scheduling	140
Functional Constraints and Limitations	142
Vendors	142
Payment Terms	143
Freight Terms	143
Currencies	144
Purchasing/Receiving	144
RMS Chart of Accounts Interface to Oracle Financials	145
RMS Currency Interface from Oracle Financials	145
RMS Error Table & Package in Interface to Oracle Financials	146
RMS GL Interface to Oracle Financials	147
RMS Interface to Oracle Financials – Initial Setup	150
RMS Purchasing Interface to Oracle Financials	152
RMS Receiving Interface to Oracle Financials	154
RMS Terms Interface from Oracle Financials	155
RMS Vendor Interface from Oracle Financials	156
Oracle Financials-Specific Loading Process Batch Program	157
RMS Invoice Matching to Oracle Financials	158
HTS	159
Functional Overview	159
Module Flow and Scheduling	159
Inventory Adjustments	160
Functional Overview	160
Module Flow and Scheduling	160

Invoicing	162
Functional Overview	162
Module Flow and Scheduling	163
Item Maintenance	164
Functional Overview	164
Module Flow and Scheduling	164
Item Reclassification	165
Functional Overview	165
Module Flow and Scheduling	165
Layaway	166
Functional Overview	166
Module Flow and Scheduling	166
Letter of Credit	167
Functional Overview	167
Module Flow and Scheduling	167
Masks	169
Functional Overview	169
Module Flow and Scheduling	169
Open to Buy Maintenance	170
Functional Overview	170
Module Flow and Scheduling	170
Organization Hierarchy	172
Functional Overview	172
Module Flow and Scheduling	172
Pricing (Retail and Cost)	173
Functional Overview	173
Module Flow and Scheduling	175
Promotions	176
Functional Overview	176
Module Flow & Scheduling	177
Purchase Order	179
Functional Overview	179
Module Flow and Scheduling	180
Receiving (P.O)	182
Functional Overview	182
Module Flow and Scheduling	184

Replenishment.....	186
Functional Overview.....	186
Module Flow & Scheduling.....	199
Retek Forecasting.....	201
Functional Overview.....	201
Module Flow and Scheduling.....	205
Return to Vendor.....	208
Functional Overview.....	208
Module Flow and Scheduling.....	208
Sales Posting.....	210
Functional Overview.....	210
Module Flow and Scheduling.....	210
Stock Counts.....	212
Functional Overview.....	212
Module Flow and Scheduling.....	214
Stock Ledger.....	216
Functional Overview.....	216
Module Flow and Scheduling.....	218
Supplier Maintenance.....	220
Functional Overview.....	220
Module Flow and Scheduling.....	220
System Administration and Control Files.....	221
Functional Overview.....	221
Module Flow and Scheduling.....	221
Tickets/Labels.....	222
Functional Overview.....	222
Module Flow and Scheduling.....	222
Transfers.....	223
Functional Overview.....	223
Module Flow and Scheduling.....	227
Transportation.....	229
Functional Overview.....	229
Module Flow and Scheduling.....	229
VAT Maintenance.....	230
Functional Overview.....	230
Module Flow and Scheduling.....	230

Chapter 16 - Batch Master Designs..... 231

Allocation Update Download [allocupd]	231
Advanced Shipment Notification Download [asndnld]	241
Audit Purge Process [audprg]	254
Audit Logic Information Edits [audsys]	256
Cost Event Purge [ccprg]	258
Customs Entry download [cednld]	260
Competitive Pricing Purge [cmpprg.pc]	273
Contract Maintenance and Purging [cntrmain]	275
Contract Replenishment – Type ‘B’ Contracts [cntrordb]	279
Contract A/C/D Replenishment Processing [cntrprss]	284
Carton Receiving Upload (for P.O. and Transfers) [ctniupld]	289
[dealcalc.pc]	294
[dealcls.pc]	309
[dealprg.pc]	312
[ditinsrt.pc]	315
Daily Record Deletion [dlyprg]	327
Increment Set System Date [dtesys]	329
Retek Store and Warehouse Address Download [edidladd]	331
EDI Contract Information Download [edidlcon]	336
[edidldeb.pc]	342
New and Changed P.O. Download [edidlord]	352
Sales and Stock Activity Report Download [edidlprd]	364
EDI Purge [ediprg]	371
Supplier Order Acknowledgements and Changes [ediupack]	373
New and Changed Supplier Address Upload [ediupadd]	384
ASN Upload [ediupasn]	388
Supplier Availability for Contracts Upload [ediupavl]	396
New and Changed Upload from Supplier [ediupcat]	400
ediupinv.pc	409
Forecasted Unit Sales Upload [fcsilupld]	424
Forecasting/Replenishment [Fcstprg]	427
Forecasted Unit Sales Rollup [fcstrbld]	429
[fdaydnld.pc]	434
[fdayupld – perl]	451
[fifcoadn.pc]	454
Currency Interface [fifcuru2107]	455
[fifglnd1107.pc]	460
[Fifglnd2107.pc]	466
[Fifglnd3107.pc]	474
[fifinvcu107.pc]	489
Generic upload program [fifpdlp107.pc]	502
Terms Interface [fiftrmu2107.pc]	504
Vendor Interface [fifvndu2107.pc]	508
Fashion Issues Data Download [fisdnldf]	510
Staple Issues Data Download [fisdnlfs]	515
Fashion Merchandise Hierarchy Download [fmednldf]	520
Staple Merchandise Hierarchy Download [fmednlfs]	534
Organizational Hierarchy Download [forgdnld]	546

Fashion Sales Data Download [fsadnldf].....	552
Staple Sales Data Download [fsadnlds]	557
[ftmednld.pc].....	562
Forecast Warehouse Download [fwhdnld].....	565
Gcdupld.pc	567
[posgpdld.pc].....	573
Sales History Rollup by Department, Class, and Subclass [hstbld].....	576
Purge Sales History [hstprg]	584
Weekly Stock on Hand and Retail Value Update for Fashion SKU/Store [hstwkfst]	586
Weekly Stock on Hand and Retail Value Update for Fashion SKU/Warehouse [hstwkfwh]	
.....	590
Weekly Stock On Hand and Retail Value Update for Staple SKU/Store [hstwksst].....	593
Weekly Stock on Hand and Retail Value Update for Staple SKU/Warehouse [hstwkswh]	
.....	597
Upload Customs tariff files [htsupld].....	600
Inventory Adjustment Purge [invaprg]	624
Stock Adjustment Update [invaupld].....	626
Close Unmatched Shipments [invclshp]	633
Invoice Post [invcpst].....	635
Automatic Invoice Matching [invmatch]	643
[invprg.pc].....	646
Item Download [itemdnld].....	649
Bill of Lading Purge [ladprg]	662
Layaway Purge [layprg]	664
Letter of Credit Download [lcadnld].....	667
[lclrbld.pc].....	698
Letter of Credit Amendment Download [lcmdnld].....	700
[lcmt700.perl].....	713
[LCMT798.perl].....	731
[lcup798.pc]	741
Letter of Credit Upload [lcupld]	746
Bill of Lading Upload [lifbolup]	754
Inventory Adjustment Conversion [lifinvup].....	763
Receiving Conversion [lifrcvup]	768
Return to Vendor Upload [lifrtvup]	775
Stock Upload Conversion [lifstkup].....	781
Location Download [locdnld]	784
[onordext.pc]	790
[orddsent.pc]	805
Purchase Order Purge [ordprg]	818
Purchase Order Information Written to Order History Tables [ordrev].....	827
Retail Price Change on Purchase Orders [ordupd].....	842
Outstanding Order Export Download File [otbdlord]	846
Stock Ledger Data Download [otbdlsal].....	854
Open to Buy Download [otbdnld].....	868
Open To Buy Purge [otbprg].....	878
Accept Forward Limit Percentages Upload [otbupfwd]	880
New Budget Data And Budget Adjustments [otbupld].....	889
Clearance Pricing POS Download [pccdnld].....	899
Retek Clearance Pricing Update [pccext]	902
Clearance Event Deletions [pccprg].....	906

Clearance Reset Pricing POS Extract [pccrdnld]	908
Retek Clearance Reset Price Update [pccrext]	912
Price Change POS Extract [pcdnld]	916
Retek Price Change Update [pccxt]	920
Price Change Impact [pcimpc]	927
Price Change Overlap [pcovrl]	929
Promotion Overlap Analysis [pcovrlpq]	931
Purge Price Changes [pcprg]	933
[pctrandn.pc]	935
[pctranex.pc]	941
Class Level Planned Sales Units Acceptance [plncupld]	948
Department Level Planned Sales Units Acceptance [plndupld]	954
Subclass Level Planned Sales Units Acceptance [plnsupld]	959
POS Configuration Download [Poscdnld.pc]	965
POS Download [posdnld]	984
[posgpdlld.pc]	990
POS Upload [posupld]	993
Inbound Work Order Download [powodld]	1013
Pre/Post Functionality for Multi-Threadable Programs [prepost]	1020
[primpdep.pc]	1026
[primpsku.pc]	1028
Promotion Price Extract and Download [prmext]	1031
Promotions Purge [prmprg]	1036
Item Level Promotions Explode [prmxpld]	1038
Promotions Download to Output File [promdnld]	1041
Automatic Shipment Extraction [rcvext]	1047
P.O. Receipt Transactions Upload [rcvupld]	1049
Receipts SQL Loader [receipts_upload.ctl]	1063
Item Reclassification [reclsdy]	1065
Item Requisition Extraction [reqext]	1071
RMS to RLS download conversion [rms2rls]	1084
Maximum Stock Levels for Fashion Items [rpladjf]	1087
Maximum Stock Levels for Staple Items [rpladj]	1090
Replenishment Attribute Update [rplatupd]	1093
Replenishment Order Build [rplbld]	1107
Vendor Replenishment Extraction [rplext]	1119
Automatic Replenishment Order Maintenance [rplprg]	1129
Replenishment [rplrsprg.pc]	1133
Return to Vendor SQL Loader [rtv_upload.ctl]	1135
Purge RTV [rtvprg]	1137
Return to Vendor SQL Loader [rtv_upload.ctl]	1139
[saaldnld.pc]	1141
[saescheat.pc]	1148
[saexpach.pc]	1156
[Saexpgl107.pc]	1170
Sales Audit Export to Invoice Matching [saexpim.pc]	1179
[saexpsfm.pc]	1196
[saexpuar.pc]	1204
[saimpadj.pc]	1212
Stock Ledger Append [salapnd]	1217
Daily Stock Ledger Processing [saldly]	1219

End of Half Stock Ledger Processing [saleoh]	1222
Stock Ledger & Budget Tables Insert [salins]	1225
Monthly Stock Ledger Processing [salmth]	1230
Purge Stock Ledger Transactions [salprg]	1237
Stock Ledger Stage [salstage]	1239
Weekly Stock Ledger Processing [salweek]	1242
[samastersfm.pc]	1255
[sapreexp.pc]	1258
[sapurge.pc]	1264
Stored Procedures / Shared Modules (Maintainability)	1266
[sarules.pc]	1272
[sastdycr.pc]	1274
[satotals.pc]	1276
Retek Supplier Cost Changes Update [sccext]	1278
Store Ship Schedule Purge [schedprg]	1287
Location Security Rebuild [slocrbld]	1289
[soutdnld.pc]	1296
Product Security Rebuild [sprdrbld]	1304
Stock Count Shrinkage Update [stkdly]	1311
Purge Stock Count [stkprg]	1315
Stock Count Snapshots Update [stkupd]	1317
Upload Stock Count Results [stkupld]	1320
Stock Count Stock on Hand Updates [stkvar]	1329
Stock Count Explode to SKU/Location for Stores [stkxplst]	1334
Stock Count Explode to SKU/Location for Warehouses [stkxplwh]	1337
Stock Ledger Extract [stlgdnld.pc]	1341
Store Add [storeadd]	1350
Supplier Constraint Scaling [supenstr.pc]	1354
Supplier download [supdnld]	1357
Sup_Data Amount Repository [supmth]	1362
Zone Security Rebuild [szonrbld]	1365
Size Ratio Build [szrtbld]	1369
SKU/Location Ticket Output File [tcktdnld]	1374
[tifposdn.pc]	1381
Transportation Upload [tranupld.pc]	1390
Stock Allocation Transfer Record Purge [tsfalprg]	1404
Combine Warehouse Originated Transfers [tsfcomb]	1415
Transfer to DC Download [tsfdcdld]	1423
[tsfdeld.pc]	1429
Approved Warehouse Transfers Download [tsfdnld]	1433
Transfer Receipts Upload [tsfiupld]	1452
Transfer Shipments Upload [tsfoupld]	1467
P. O. Receipt, Transfer Receipt, and Carton Receipt Parse [tsfparse]	1488
Transfer Record Purge Transactions [tsfprg]	1501
Reserve Inventory for Stock Allocation [tsfresv]	1510
[txrposdn.pc]	1516
[Txrtupld.pc]	1522
VAT Rate Maintenance [vatdlxpl]	1527
Vendor Replenished Order Build [vrplbld]	1530
Wastage Adjustment [wasteadj]	1542
Warehouse Store Assignment [whstrasg]	1545

Chapter 17 - Overview of RMS Batch Programs 1549

Batch Overview	1549
Restart and Recovery	1549
Data Model Discussion	1555
Physical Set-up.....	1555
Table & File-Based Restart/Recovery	1556
API Functional Descriptions	1559
Multi-Threading	1568
Threading description	1568
Threading Function for Query-based	1568
Restart View for Query-based.....	1570
Thread Scheme Maintenance:	1572
Batch Maintenance:.....	1573
Scheduling & Initialization of Restart Batch	1573
Pre/post processing	1574
Query-based Commit Thresholds	1574
Array Processing	1574
Input / Output Formats	1574
General Interface Discussion	1574
Detail only files.....	1575
EDI.....	1578
Interfaces Supported by RMS	1579
Interface File Layouts	1579

Chapter 1 - Introduction

Welcome to the Retek Enterprise 9.0 Operations Guide. The guide is designed to give you information about the “backend” of Enterprise 9.0’s core merchandising transaction products. It does not show you how to use any of the products; rather it lets you see how the merchandising products work behind the scenes. The focus is on how data is managed; in other words, how data flows and the batch programs that facilitate the movement of data.

What This Guide Shows You

This guide shows you the flow of data:

- Within Retek Merchandising System (RMS), Retek Enterprise 9.0’s core product
- Between RMS and other merchandising transaction system (MTS) products, including:
 - Active Retail Intelligence (ARI)
 - Invoice Matching
 - Competitive Shopping
 - Sales Audit
 - Store Operations
 - Trade Management
- Between RMS and MTS products and these Retek Enterprise 9.0 standalone products, including:
 - Retek Data Warehouse (RDW)
 - Retek Demand Forecasting (RDF)
 - Retek Distribution Management (RDM)
 - Retek Predictive Planning (RPP)

- Between Retek Enterprise 9.0 products and third-party software applications and entities, including:
 - Point-of-sale (POS) systems
 - Financial applications such Oracle Financial Application version 10.7
 - Suppliers
 - Account Clearinghouse (ACH)
 - Driscoll Universal Account Reconciliation (UAR)
 - Oracle Site Fuels Management
 - Banks
 - Import trading partners, brokers, and transportation providers

What's in This Guide

The guide contains three kinds of information:

1. High-level overviews of data from the perspective of RMS batch programs (or modules):
 - Posting Sales Data to RMS
 - RMS-electronic data interchange (EDI)
 - RMS-Invoice Matching
 - RMS-Oracle Financial Application 10.7
 - RMS-RDF
 - RMS-RPP
 - RMS-RDM
 - RMS-RDW Interface
 - RMS-Retek Trade Management (RTM)
 - Retek Sales Audit (ReSA)
 - Updating the point-of-sale system (POS) from RMS

2. Functional overviews of batch program operations and the movement of data. These overviews are somewhat more technical in nature than the high-level overviews. Functional overviews describe batch processing for specific component areas of the merchandising transaction system products.
3. Detailed designs of batch modules. These designs describe how an individual batch module works and the database tables that it affects. Detailed designs are the most technical documents in the guide.

What's Not in This Guide

Descriptions of batch processing for Retek Data Warehouse, Retek Distribution Management, Retek Demand Forecasting, Retek Predictive Planning are not in this guide. High-level overviews of the kinds of data that flow between RMS and each of these products are in this guide; however, they focus on RMS batch modules responsible for presenting data to those applications and processing data received from those applications. At the end of this introduction, you can find a list of operation guides for these three Retek products that will provide you with further information.

Who This Guide is Written for

Anyone who has an interest in better understanding the inner workings of the Retek merchandising transaction system can find valuable information in this guide. There are two audiences in general for whom this guide is written:

- Business Analysts—Those who are looking for information about merchandising transaction system batch modules that allow them to compare programmatic dataflows to a customer's business processes, or workflows.
- System Analysts and Database Administrators—Those who are looking for information about batch modules singly or in relation to others and for those who need information for planning a batch-processing schedule.

Where You Can Find More Information

You can find more information about Retek Enterprise 9.0 products in these resources:

- RMS online help
- RMS Getting Started Guide
- Data Model, vol. 1 and 2
- RDW manuals
- RDM manuals
- RDF manuals

How to Use This Guide

As described earlier, this guide offers you high-level overviews, functional overviews, and individual batch module designs.

Read this...	If you want to know about...
Dataflow overview...	...the kinds of data that flow between programs or components of programs and the batch modules used and their dependencies on other modules in a batch schedule
Functional overview...	...what batch programs are included in a functional area
Design document...	...details of how an individual batch module works including database tables referenced by the batch module

Other Good Things to Know

- Not all functional areas have batch modules associated with them.
- Dataflow overviews of RDW, RDM, RDF, and RPP only describe RMS (MTS) batches that process data to those apps. or from those apps.

Chapter 2 - Glossary of Abbreviations, Acronyms, Terms

The overview sections of this guide contain a number of abbreviations, acronyms, and terms that are part of the language used within Retek development. This section is intended as a useful guide for interpreting some of this language.

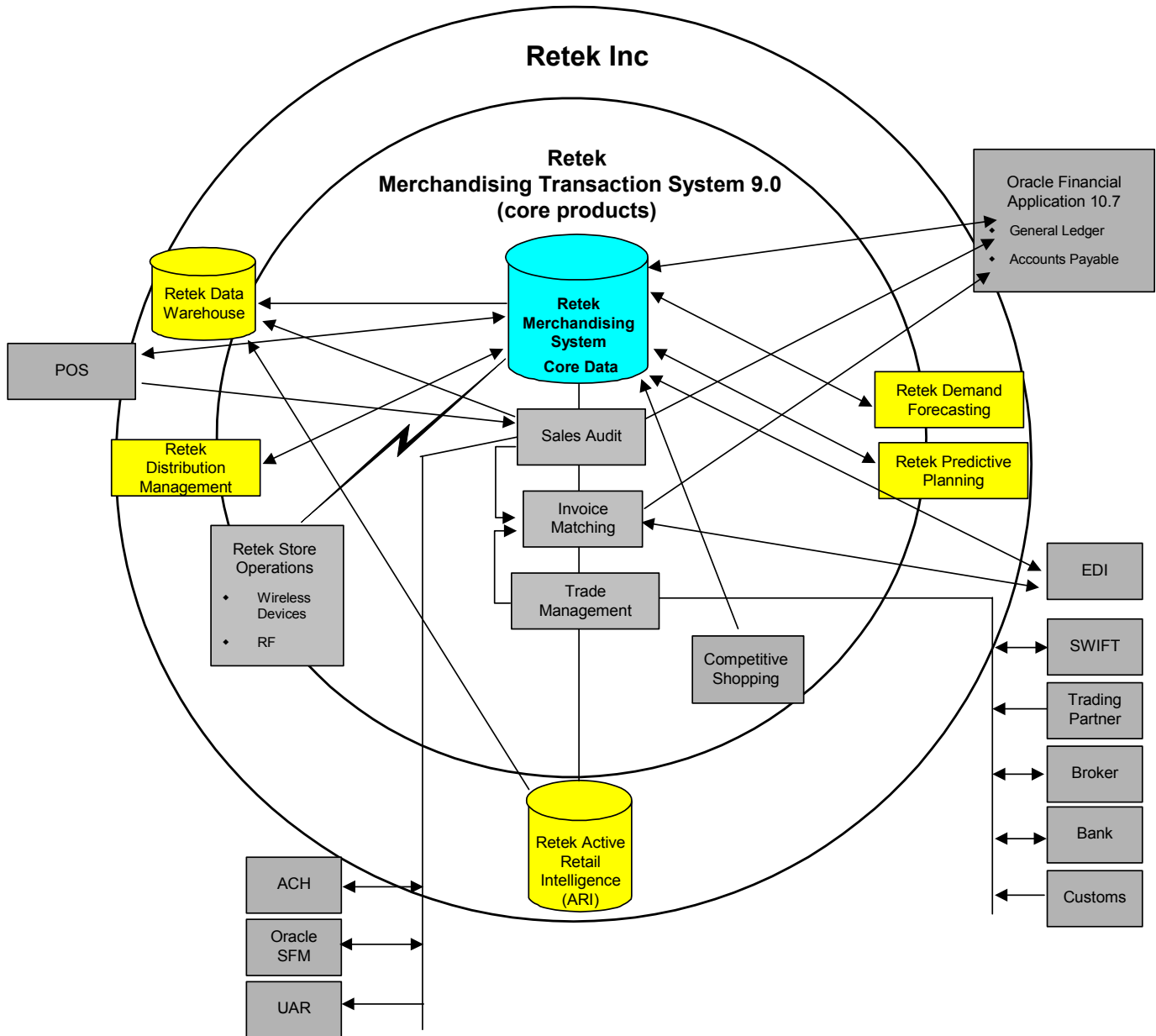
ACH	account clearing house
ARI	Active Retail Intelligence
array processing	A method of defining and processing groups of records, rather than processing one at a time. Array processing improves batch processing by limiting the number of interactions with the database. See also: logical unit of work, restart and recovery, and restart and recovery
ASN	advanced shipment notice (from supplier)
commit (commit max counter)	The number of records that a batch module processes before it stores (commits) the data to the database. A commit max counter for each batch program is stored in a table and used in the Retek restart and recovery process to determine when commits should take place. See also, restart and recovery, logical unit of work.
cursor	Currently selected set of records.
EDI	electronic data interchange
full disclosure	The process by which ReSA adjusts previously exported values fully backing out old values before inserting new values. Contrast full disclosure with a simple adjustment.
HTS	harmonized (U.S. Customs) tariff schedule
logical unit of work	The variables in a batch module used to order data to be processed to the database. The variables define a transaction to be processed. When the batch module runs, the logical unit of work is considered in conjunction with the commit max counter to determine commit points. For example, 10,000 SKU/store combinations are processed in order and then committed to the database. The benefit of using the logical unit of work to order data is that if the batch module encounters an error or otherwise stops running, it can return to the last commit point before continuing processing. Previously committed data does not need to be reprocessed. See also: restart/recovery, commit, multi-threading.

multi-threading	Running a batch module on separate groups of data simultaneously. For example, multiple instances of the batch module POSUPLD can run concurrently to allow the processing of sales transactions for more than one store at one time. Multi-threading reduces the total amount of processing time. See also: commit, logical unit of work, restart and recovery.
POS	point of sale system
POSU file	Point of sale upload file created by the customer in a Retek format usable in RMS's point-of-sale upload process. This customer-generated file contains the customer's sales transaction data rolled up by SKU for the store location for one day. If the customer regularly polls its POS system during the day, more than one POSU file is created for that store day.
RDM	Retek Distribution Management
RDW	Retek Data Warehouse
ReSA	Retek Sales Audit
restart and recovery	A Retek process that allows a batch module to recover from a failed process by restarting at the point in its data set where it failed. Restart and recovery improves processing efficiency by avoiding the reprocessing of data after a failure. See also: logical unit of work, cursor, and multi-threading.
RMS	Retek Merchandising System
RTLOG	Customer sales transaction data transferred from the POS in a file format that can be used in ReSA. The customer is responsible for converting its transaction logs to RTLOGs.
RTM	Retek Trade Management
SFM	Oracle Site Fuels Management—ReSA can interface SFM.
SKU	stock keeping unit
store day	Specific to ReSA, store day pertains to all sales transactions for one business day at one store

SWIFT	Society for Worldwide Interbank Financial Telecommunications—a standard file format to be used within RTM for future letter of credit functionality. Currently under development.
TLOG	Customer's transaction log in a format specific to the customer's point of sale system
UAR	Universal Account Reconciliation—a product developed by a company called Driscoll. ReSA can export data to UAR.
VAT	value added tax

RETEK ENTERPRISE 9.0 Dataflow Overview

(Arrows indicate flow of data.)



Note: Programs and entities outside of Retek Retail Solutions circle are external to Enterprise 9.0.

Chapter 3 - Interface Dataflow Overviews

RMS Batch Modules for Retek Trade Management

Retek Trade Management (RTM) automates international import transaction data. There are six components of RTM:

- Customs entry
- Harmonized tariff schedule
- Letter of credit
- Transportation.
- Actual landed costs
- Obligations

Four of these components—customs entry, Harmonized Tariff Schedule, letter of credit, and transportation—have batch-processing modules that facilitate the flow of data between RTM and external applications and files. This document describes these batch modules and the kinds of data that they process.

New in RTM 9.0

RTM 9.0 adds:

Integration with Retek Invoice Matching and Oracle Financial Application 10.7 — Obligations and customs entry costs can be approved for payment and entered into Retek Invoice Matching (RIM) where they are transferred to Oracle Financial Application 10.7 Accounts Payable for payment.

Letter of Credit Data Downloads and Upload via S.W.I.F.T. — RTM supports the downloading of letter of credit applications and amendments to banks and the uploading of drawdowns, bank fees, and confirmations from banks in standard S.W.I.F.T. file formats.

Batch Modules

RTM uses seven batch modules among these functional areas:

Customs Entry. The customs entry module CEDNLD.PC downloads customs entry data from RTM to individual custom broker IDs. Customs Entry Download. Some of the information transferred by this batch module includes:

entry	importer	surety bond	merchandise
currency	exchange rate	vessel	port
visa	invoice		

CEDNLD.PC outputs one flat file (text file) for each broker ID.

Harmonized Tariff Schedule (HTS). The one HTS batch module — HTSUPLD.PC, Harmonized Tariff Schedule Upload—processes a file containing the most recent United States Customs tariff schedule to RMS tables. The module uploads both the initial entry of the schedule and all updates, as they become available.

Letter of Credit. Letter of credit batch modules process letter of credit applications and amendments to banks and to upload confirmations, drawdown notifications, and related information from banks. Letter of credit downloads and uploads data in an internationally recognized standard format called S.W.I.F.T. (Society for Worldwide Interbank Financial Telecommunications).

The Letter of Credit Application Download module, LCADNLD.PC, downloads approved letter of credit applications to banks. Run LCADNLD.PC before the script LCMT700.Pperl that converts the applications from a Retek file format to the S.W.I.F.T. (MT 700) format.

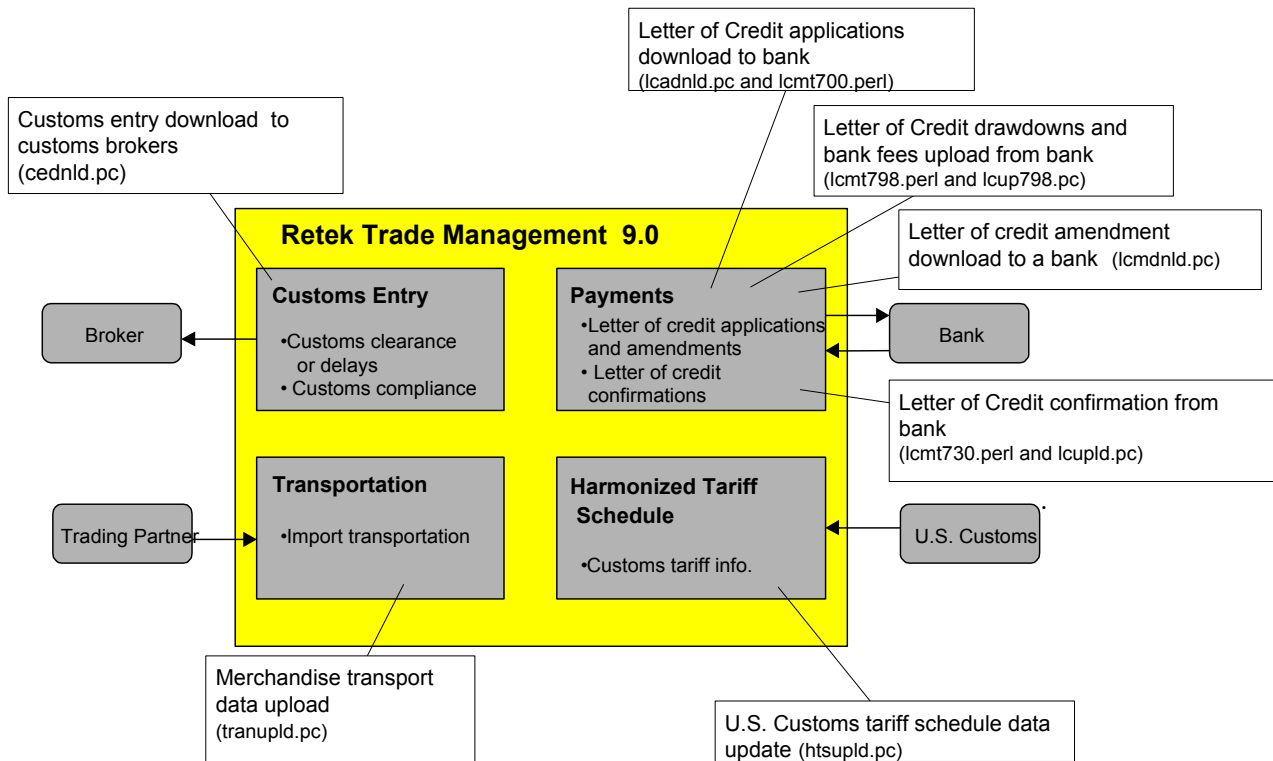
RTM uses one process to upload letter of credit drawdowns and bank fees. It uses a second to upload letter of credit confirmations. The script LCMT798.Pperl converts drawdowns and bank fees data from a S.W.I.F.T. file format to a Retek format. The batch module LCUP798.PC then writes the converted data to the table LC_ACTIVITY.

The script LCMT730.Pperl converts letter of credit confirmations from a S.W.I.F.T. format (MT730) to a Retek formatted file. The batch module LCUPLD.PC then writes the converted data to the table LC_HEAD. [IS THIS TABLE CORRECT]

The last Letter of Credit batch module is called LCMDNLD.PC for Letter of Credit Amendment Download. The module downloads amended letter of credit information to a bank, again in the S.W.I.F.T. format.

Transportation. This RTM component has one batch module, Transportation Upload, TRANUPLD.PC. This module uploads data from trading partners about the transport of merchandise from the manufacturing site through customs clearance.

The following graphic describes RTM functionality and batch processing.



Batch Modules at a Glance

This table summarizes the Retek Trade Management batch modules that are associated.

Retek Trade Management Batch Modules		
Batch Module (or Perl script) Program	What It Does	When to Run It
cednld.pc	Transfers data about customs to a broker	N/A
prepost.pc with the argument htsupld_pre()	Truncates the table populated by htsupld.pc mod_order_item_hts before htsupld.pc runs again for the newest tariff schedule	Run before htsupld.pc
htsupld.pc	Processes a file containing the most recent United States Customs tariff schedule to RMS tables	Run after prepost.pc with the argument htsupld_pre()
lcadnld.pc	Downloads approved letter of credit applications to a bank in a Retek format	Run before the script lcmt700.perl
lcmt700.perl	Script that converts approved letter of credit applications from a Retek format to the S.W.I.F.T. (MT700) format	Run after lcadnld.pc
lcmt798.perl	Script that converts letter of credit drawdowns and bank fee data from a S.W.I.F.T. (MT798) format to a Retek format suitable for uploading into RMS	Run before lcup798.pc
lcup798.pc	Writes data from the file converted by lcmt798.perl to the RMS table lc_activity	Run after lcmt798.perl
lcmt730.perl	Script that converts letter of credit confirmations from a S.W.I.F.T. (MT730) format to a Retek format suitable for uploading into RMS	Run before lcupld.pc
lcupld.pc	Writes data from the file converted by lcmt730.perl to the RMS table lc_head	Run after lcmt730.perl
lcmdnld.pc	Downloads amended letter of credit information to a bank	N/A
tranupld.pc	Uploads data from trading partners about the transport of merchandise from the manufacturing site through customs clearance	N/A

Chapter 4 - Understanding EDI Batch Modules

Electronic Data Interchange (EDI) is a convenient method by which you and suppliers can transfer information back and forth. Data transferred typically includes contracts and purchase orders, acknowledgements and confirmations, advance shipment notifications (ASN), merchandise availability, and invoices. You need to set up your suppliers in Retek Merchandising System and then select EDI options for as many transaction types as a supplier supports and uses.

RMS lets you take advantage of EDI and supplies you with ten (10) batch processing modules that update EDI-related information between RMS and your database. Each of these ten modules are summarized in a table. Detailed information about each module is in the Retek Enterprise 3.0.2 Operations Guide.

Because the format of EDI files is not directly supported by RMS, customers need to use an EDI translation product to convert EDI data to and from an RMS flat file format.

Batch Modules at a Glance

This table shows you an overview of RMS batch modules that are associated with EDI transaction types. See individual batch module designs for detailed descriptions.

Electronic Data Interchange (EDI) Batch Modules			
Batch Module Program	EDI Transaction Number Supported	What It Does	When to Run It
ediupavl.pc		Uploads the supplier availability schedule.	Daily, Phase I.
edidlcon.pc	850	Downloads contract information. Only approved contracts are processed. Also you must select the EDI Contract field on the Contract Header Maintenance window in order to use this function.	Daily, Phase IV.
ediupasn.pc	856	Uploads advance shipment notification (ASN) received from the supplier, containing orders and items. ASNs are used to create shipment records.	Daily, Phase I.
ediupcat.pc	832	Uploads new or modified item information, including item cost changes. The tables edi_new_item and edi_cost_change are modified.	Daily, Phase II.
ediupadd.pc	838	Uploads supplier address changes. As many as five different supplier address types can be modified.	As needed.

Electronic Data Interchange (EDI) Batch Modules			
Batch Module Program	EDI Transaction Number Supported	What It Does	When to Run It
ediupack.pc	855	Uploads supplier acknowledgements and changes received by supplier from retailer, including:	As needed, Phases I, II, or III.
edidladd.pc	838	Download to a supplier a list of retailer's store and warehouse addresses, both new or modified. You must select the Create EDI Address Catalog field on the System Parameter Maintenance window in order to use this function.	Daily, Phase 4.
edidlprd.pc	852	Download a sales audit summary containing sales details, current stock on hand by location, and stock quantities currently in transit for SKUs supplied by that supplier.	Daily, Phase 4.
edidlord.pc	850 and 860	Downloads new and modified purchase orders. If applicable, only the most recent order and allocation information is sent. You must select the EDI PO field on the PO Header Maintenance window in order to use this function.	Daily, Phase 4, or as needed.
ediprg.pc	N/A	Purges new items or cost changes that have been rejected online. Items are purged based upon the number of days set by the administrator.	As needed.

Chapter 5 - The RMS and Retek Demand Forecasting 9.0 Interface

Retek Demand Forecasting (RDF) version 9.0 forecasts sales and warehouse issues that Retek Merchandising System (RMS) version 9.0 can use in its replenishment and mask modules. RMS uses several batch processing modules that are designed to send RDF data and structures that it incorporates in its sales forecasts. Other RMS batch modules upload and process RDF sales forecasts that RMS automated replenishment and mask modules can use. This document describes RMS data and batch modules involved with the RDF interface as well as the process by which forecasted sales and warehouse issues enter RMS.

Here is the order of processes in the RMS and RDF interface:

1. Set up domains in RMS.
2. Download to RDF:
 - a. Store and warehouse (organizational hierarchy) data
 - b. Staple and fashion (merchandise hierarchy) data
 - c. Time hierarchy (4-5-4 fiscal year calendar)
 - d. Domain-specific SKU sales history and SKU warehouse issue history data
 - e. Stock outs (locations that have no stock of a particular SKU)
3. Upload RDF sales and warehouse issues forecasts and roll up the sales forecasts from the SKU/store/week level to the subclass, class, and department/store/week level. Be sure to read the next section for enhancements.

New to the RMS 9.0-RDF 9.0 Interface

Daily Sales Forecasts—In addition to the existing weekly sales forecasts, RMS 9.0 now sends RDF 9.0 daily sales transactions and warehouse issues and in return receives daily sales forecasts. Also, RMS sends RDF daily stockouts (stock outages at a store)-on-hand outages. Because certain areas of RMS do not use daily forecasts, daily and weekly forecasts exist in parallel. RMS's replenishment and scaling modules use the daily forecasts whenever they are available. If the daily forecasts do not cover the time period needed by replenishment, it will revert back to its current method of using 1/7 of the weekly forecast.

Calendar download—RMS downloads the customer's 4-5-4 calendar to RDF, thereby no longer requiring RDF to maintain its own calendar. The calendar adjusts for fiscal year start dates.

Domains and their Purpose

The use of domains is a method by which RDF is able to more easily manipulate data. They are RMS SKUs divided into logical sales forecast groups. RMS sends RDF the sales transaction histories for SKUs that are marked as 'forecastable' within RMS by dividing them into these domains. RDF then loads the domains into individual instances of the database within RDF before creating daily and weekly sales forecasts.

In RMS a domain is a logical subset of data for which you would want forecasted sales and warehouse issues. Domains are established by department, class, or subclass. Because many companies choose to examine data at the department level, this is one logical choice for a domain, but not the only one. Domains are set up in RMS on the domain table and on the domain_dept, domain_class, and domain_subclass tables.

Data Downloaded to RDF

RMS prepares and transfers (downloads) the following data to RDF:

- Complete merchandise hierarchy specific to each domain
- Complete organizational hierarchy copied to each domain
- The RMS calendar adjusted for the customer's fiscal year (used by each domain)
- Stockouts (stock-on-hand outages at stores)
- Daily and weekly historical sales data for all forecastable SKU items in a domain

Batch Modules That Download RMS's Organizational and Merchandise Hierarchies

Two batch modules download RMS's organizational hierarchy for stores and warehouses to RDF, two other batch modules download the merchandise hierarchy, and a fifth batch module downloads the calendar (time hierarchy):

- Organization Hierarchy (stores) Download: FORGDNLD.PC
- Warehouse Download: FWHDNLD.PC
- Staple Merchandise Hierarchy Download: FMEDNLDS.PC
- Fashion Merchandise Hierarchy Download: FMEDNLDF.PC
- Time Hierarchy Download: FTMEDNLD.PC

All domains share the same organization hierarchy (stores and warehouses). The merchandise hierarchy (staple and fashion SKUs) differs based upon domain.

Determine Data to Download to RDF

After you set up domains, six RMS batch modules download data to RDF, two to download sales transaction histories, two to download warehouse issue histories, one to download daily sales histories, and one to download locations that are out of stock for individual SKUs:

- Staple sales data download: FSADNLDS.PC
- Fashion sales data download: FSADNLDF.PC
- Staple issues data download: FISDNLDS.PC
- Fashion issues data download: FISDNLDF.PC
- Daily sales transaction history: FDAYDNLD.PC
- Stockout Download: SOUTDNLD.PC

Histories can be downloaded only for those items that are marked for forecasting on the win_skus table (for staple SKUs) or on the rag_style table (for fashion style SKUs).

This is the process by which history periods for SKUs are selected. The batch modules first determine the beginning date of the history period for sales transactions or warehouse issues that will be downloaded to RDF. This is done by examining an export date field on the appropriate staple or fashion by store table or staple or fashion by warehouse table, depending upon the batch module. If a date of last export exists, the batch module selects the history for all weeks greater than that date. If no date exists in the table's field, the batch module selects the entire history.

Alternatively, if the load_sales_ind field on the domain_dept, domain_class, or domain_subclass table (depending upon the selected domain_level field setting on the system_options table) displays the letter Y, all the sales history for all SKUs in the domain are selected.

After RDF has received all of this data, it can begin performing its forecasting functions. Upon completion, RDF makes sales and warehouse issue forecasts available to RMS. The last three RMS batch modules for this interface complete the work of updating RMS with forecasts.

Upload and Process RDF Sales and Warehouse Issues Forecasts

RDF makes the following daily and weekly forecast data available to RMS using two batch modules and one Perl script:

- Store sales and warehouse issues forecasts by SKU/location/week: FCSLUPLD.PC
- The weekly store sales forecasts by SKU that are rolled up within RMS to the location/week level by subclass, class, and department: FCSTRBLD.PC
- Store sales and warehouse issues forecasts by SKU/location/day: FDAYUPLD, a Perl script

RMS uploads forecast data weekly. Before it does so, it must clear the current domain data on the table because a complete set of forecasts by domain is exported each time by RDF. The batch module Forecasted Sales Purge (FCSTRPRG.PC) purges forecast tables.

The Forecasted Sales Upload batch module (FCSLUPLD.PC) then runs to upload SKU/store/week level sales forecasts along with warehouse issues forecasts. Make sure that you run this module once for each domain uploaded from RDF. This batch creates ASCII text files that populate RMS' item_forecast table using the SQL*Load process.

The batch module Forecasted Sales Rollup (fcstrbld.pc) rolls up forecasted store sales to the subclass/store/week level (subclass_sales_forecast table), class/store/week level (class_sales_forecast table), and the department/store/week level (dept_sales_forecast table). These sales forecast tables are then available to be accessed by RMS' automated replenishment module, which accesses the item forecast table; and RMS' mask module, which accesses the item, subclass, class, and department forecast tables.

Finally, for the daily sales forecast upload, the Perl script FDAYUPLD runs to populate the daily forecast table, once for each RDF domain. The script accepts ASCII text files and uses the SQL*Load process to load the RMS table. Prior to this process the script clears the tables of previously loaded data.

This figure summarizes the data that the RMS batch modules transfer to RDF and the data created in RDF that RMS batch modules upload and process.

Note: Combine RDF files before processing into RMS

Upon receiving from RDF the daily sales and daily issues forecast files for the same domain or the weekly sales and weekly issues forecast files for the same domain, you must combine the two files (the two daily ones or the two weekly ones) into one file prior to running fcslupld.pc and the fdayupld Perl script. The RMS database expects one file per domain for the daily and weekly uploads.

Failure to combine the files will result in overwriting the first file that you load with the second.

**Data Sent to RDF
Used in Forecasting**

- SKUs by staple and fashion style and their supplier information by domain
- Daily and weekly sales history data for SKUs by domain
- Daily and weekly warehouse issues data for SKUs in each domain
- Time hierarchy adjusted for different fiscal year
- Complete organization hierarchy
- Complete merchandise hierarchy
- Stockout at stores

**Data Created in RMS
from RDF Sales Forecasts**

- Weekly store sales and warehouse issues forecasts (SKU/store/week)
- Store sales forecasts rolled up to the store/week level by subclass, class, and department
- Daily sales and ware house issues forecasts (SKU/store/day)

Dataflows between RMS and RDF

Tables of Batch Modules

The following tables list the RMS batch modules that are involved with preparing and downloading data to RDF and with uploading and processing forecast data from RDF.

RMS Batch Modules for Transferring Data to RDF			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
forgdnld.pc	Downloads store location data from RMS's organizational hierarchy to RDF	Ad hoc	N/A
fwhdnld.pc	Downloads warehouse location data from RMS's organizational hierarchy to RDF	Ad hoc	N/A
fsadnlds.pc	Downloads forecastable staple item SKUs and sales transaction history data for a specific week ending period	Phase 4	N/A
fsadnldf.pc	Downloads forecastable fashion style item SKUs and sales transaction history data for a specific week ending period	Phase 4.	N/A
fisdnlds.pc	Downloads forecastable staple warehouse SKU issues history for a specific week ending period	Phase 4	N/A
fisdnldf.pc	Downloads forecastable fashion warehouse SKU issues history for a specific week ending period	Phase 4	N/A
fdaydnld.pc	Download daily sales history, warehouse issues history, and stock-on-hand data	Phase 4	
soutdnld.pc	Downloads store stockouts by domain	Phase 4	

RMS Batch Modules for Transferring Data to RDF			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
fmednlds.pc	Downloads a copy of the staple merchandise hierarchy	Ad hoc.	Run before the daily inventory and sales
fmednldf.pc	Downloads a copy of the fashion merchandise hierarchy	Ad hoc.	Run before the daily inventory and sales
ftmednld.pc	Downloads the time hierarchy (4-5-4 calendar only) adjusted to the customer's fiscal year		

Note: The programs that download store sales and warehouse issues generally run in phase four (4) of the RMS batch schedule.

RMS Batch Modules for Transferring and Processing Data from RDF			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
fcstrprg.pc	Purges previously created forecast tables.	Ad hoc.	Run before fcslupld.pc.
fcslupld.pc	Uploads forecasted sales and issues at the SKU/store (or warehouse)/week level.	Ad hoc.	Run after fcstrprg.pc and before fcslupld.pc
fcstrbld.pc	Rolls up forecasted sales to the subclass/store/week level (subclass_sales_forecast table), class/store/week level (class_sales_forecast table), and the department/store/week level (dept_sales_forecast table).	Ad hoc.	Run after fcslupld.pc
fdayupld Perl script	Loads daily sales forecasts by SKU/location/date to RMS' daily_item_forecast table	Ad hoc	N/A

Note: The programs that upload generally run weekly in phase two (2) of the RMS batch schedule.

Chapter 6 - The RMS and Retek Predictive Planning 9.0 Interface

Retek Merchandising System 9.0 (RMS) interfaces Retek Predictive Planning 9.0 (RPP) in support of open-to-buy (OTB) planning. The customer can choose to send RPP stock ledger data at the subclass/location/week level either for the most current week or for a historical period. After RPP processes the RMS data, it returns OTB budget calculations.

If the customer chooses a historical period, data is sent for:

- Values — for sales, beginning of period, return to vendor, reclassification, permanent markdowns, promo markdowns, clearance markdowns, shrinkage, receipts, transfer in, transfer out;
- Costs — for , beginning of period, sales, receipts, transfer in, transfer out;
- Sales Units

If the customer chooses to download the most current week, data is sent for the same values, costs, and sales units along with on-order data for value and cost. On-order data are not applicable to the historical data download.

Note: All data sent to RPP, other than for on-order, is derived from RMS' WEEK_DATA table. For detailed information about data sent to RPP, see the RMS 9.0 Data Model along with the design document for the batch program stlgdnld.pc in the RMS 9.0 Batch Operations Guide.

The RMS-RPP interface consists of flat file (ASCII-based, 'text' files) processing between the two applications.

RMS uses several batch-processing modules that are designed to send RPP data and structures that RPP incorporates into its OTB calculations. Another RMS batch module uploads and processes RPP open-to-buy data, making it available to users online. This document describes RMS data and batch modules involved with the RPP interface and the processes used.

Here is the order of processes in the RMS and RPP interface:

1. Download to RPP:
 - a. Store and warehouse (organizational hierarchy) data
 - b. Staple and fashion (merchandise hierarchy) data
 - c. Time hierarchy (4-5-4 fiscal year calendar)
 - d. Historic data load at the subclass, location, and week level (for the historical period selection only)
 - e. Weekly transaction data at the subclass, location level from the stock ledger, along with on-order merchandise data.

2. Upload open-to-buy data from RPP.
 - a. Complete store and warehouse (organizational hierarchy) data

Descriptions of both the download and upload processes follow.

RMS Hierarchy Data Downloaded to RPP

RMS prepares and transfers (downloads) the following data to RPP:

1. Complete staple and fashion (merchandise hierarchy) data
2. The RMS calendar adjusted for the customer's fiscal year (4-5-4 fiscal year calendar only)

Two batch modules download RMS's organizational hierarchy for stores and warehouses to RPP:

- Organization Hierarchy (stores) Download: FORGDNLD.PC. This batch module includes selling space for each store.
- Warehouse Download: FWHDNLD.PC

Two additional batch modules download the merchandise hierarchy:

- Staple Merchandise Hierarchy Download: FMEDNLDS.PC
- Fashion Merchandise Hierarchy Download: FMEDNLDF.PC

One last batch module downloads the calendar (time hierarchy):

- Time Hierarchy Download (4-5-4 calendar only, adjusted to customer's fiscal year): FTMEDNLD.PC

RMS Weekly Sales Data Downloaded to RDF

RMS also sends RPP weekly transaction data that RPP uses in its OTB calculations. If the customer chooses to send data for the current week, the process involves the extraction of on-order data that RPP incorporates into OTB calculations, along with the weekly transaction data. Note that on-order data only include orders with an expected delivery date (that is, a "not before date") that is within the planning horizon. If a historical period is selected for download, on-order data are not included.

Here is the process that RMS uses to determine on-order data to download to RPP. If the order has allocation(s) attached and is a pre-marked order, the on-order reflects the amounts for the store allocations and not the amounts for the warehouse order location—that is, the amount flowing through the warehouse. If the order has allocation(s) attached and is not a pre-marked order, the on order amounts reflect the amounts for the warehouse order location and not the allocation locations. If the order has no allocations attached, the on-order amounts reflect the amounts for the order locations.

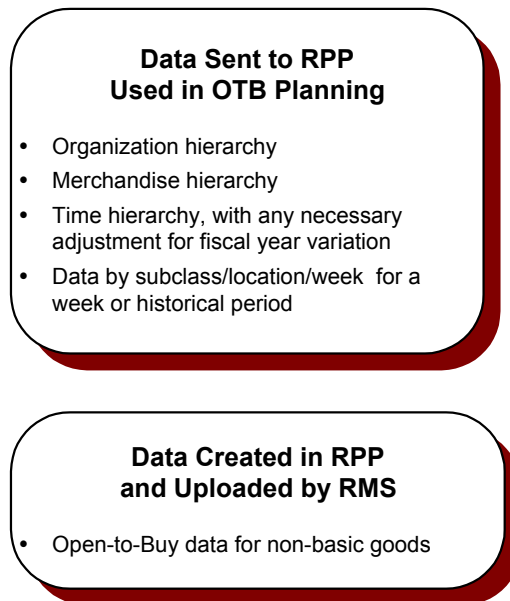
The batch module ONORDEXT.PC extracts on-order data and writes them to a temporary table. Then the batch module STLGDNLD.PC combines data from the temporary table together with weekly transactions (at subclass/location/week level) from RMS' stock ledger. After RPP has received all of this data, it can begin calculating OTB budget numbers

Upload and Process RPP Open-to-Buy Data

After calculating open-to-buy budget figures, RPP makes the data available to RMS. RMS uses its batch module OTBUPLD.PC to upload and process OTB data. In order to take advantage of RPP's open-to-buy data, RMS users need to:

- Select the OTB calculation type of 'Retail Inventory' on the Department Maintenance form within RMS, and
- Know that RPP data are only used to populate the RMS' order category of 'N,' ('Order of Non-Basic Goods'). Non-basic goods are purchased and approved manually.

The following graphic summarizes the data that the RMS batch modules transfer to RPP and the data created in RPP that RMS uploads and process.



Dataflows between RMS and RPP

Summary of Batch Modules

The following tables list the RMS batch modules that are involved with preparing and downloading data to RPP and with uploading and processing open-to-buy data from RPP.

RMS Batch Modules for Transferring Data to RPP			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
forgdnld.pc	Downloads store location data and store selling space from RMS's organization hierarchy to RPP	Ad hoc	N/A
fwhdnld.pc	Downloads warehouse location data from RMS's organizational hierarchy to RPP	Ad hoc	N/A
fmednlds.pc	Downloads a copy of the staple merchandise hierarchy	Ad hoc	
fmednldf.pc	Downloads a copy of the fashion merchandise hierarchy	Ad hoc	
ftmednld.pc	Downloads the time hierarchy (4-5-4 calendar only) adjusted to the customer's fiscal year	Ad hoc	
onordext.pc	Extracts on-order data	Weekly	
stlgdnld.pc	Sends sales data from the stock ledger at the subclass/location/week level (from RMS' week_data table)	Phase 4, weekly or ad hoc	Run after onordext.pc on a weekly basis (for the weekly download) Onordext.pc does not run for a historic download

RMS Batch Modules for Transferring Data to RPP			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
prepost.pc (with the argument onordext_pre())	Truncates the on-order temporary table before onordext.pc runs	Phase 4	Run before onordext.pc

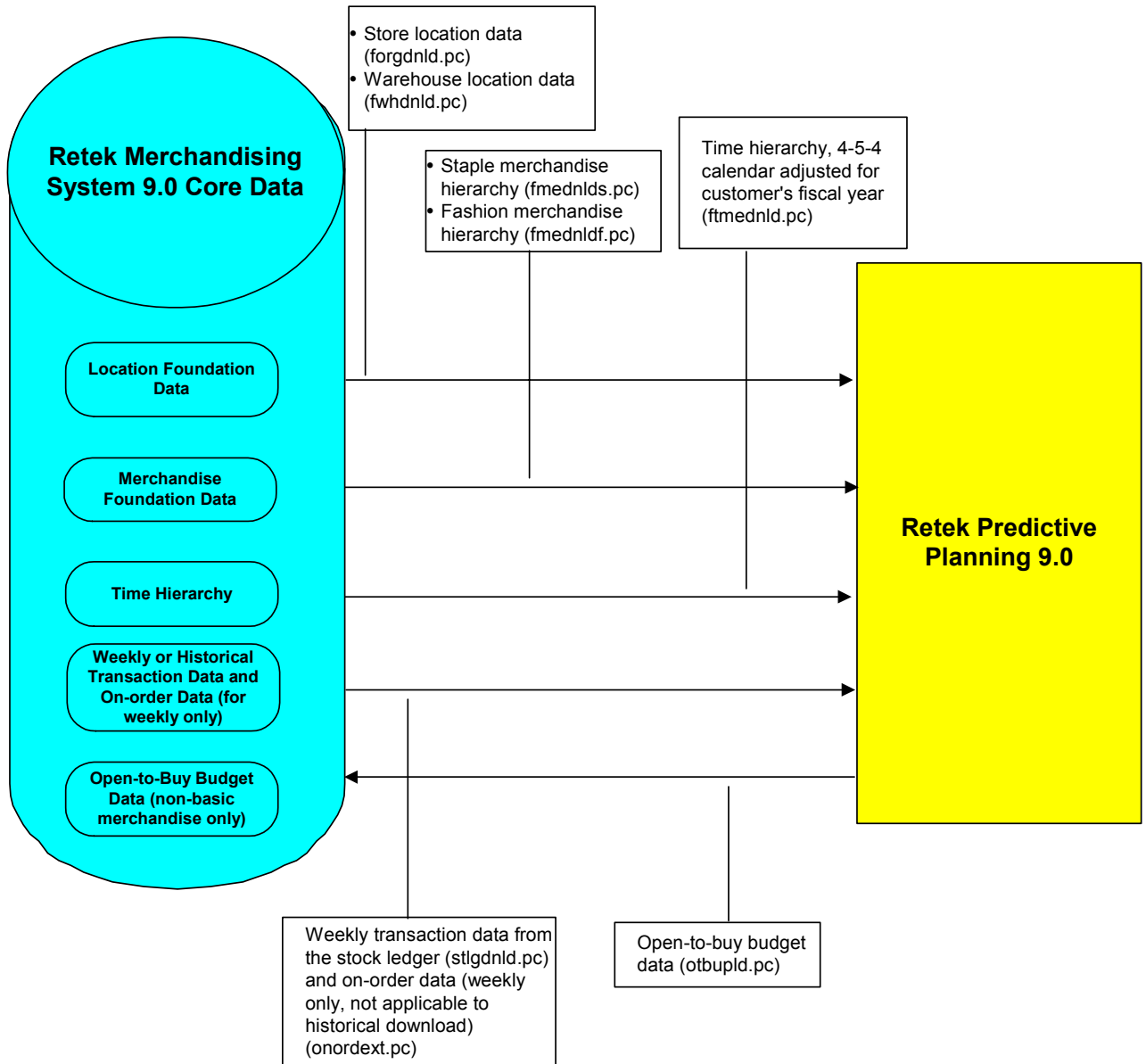
RMS Batch Module for Uploading and Processing Data from RPP			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
otbupld.pc	Uploads open-to-buy budget data for non-basic merchandise	Ad hoc or Phase 2 weekly	

RETEK ENTERPRISE 9.0

Dataflow Overview:

Retek Merchandising System and Retek Predictive Planning

Arrows indicate flow of data. RMS batch modules listed in parentheses.



Chapter 7 - The RMS 9.0 and Retek Distribution Management 9.0 Interface

The Retek Merchandising System (RMS) interfaces Retek Distribution Management (RDM), Retek Inc.'s warehouse and distribution center management application, in three functional areas:

- Organization and merchandise hierarchies.
- Purchasing
- Movement of inventory.

This document describes the data that flows between these two applications and the RMS batch processing modules involved in moving that data. Also included are descriptions of RMS batch modules that convert file data uploaded from RDM for RMS files.

RMS Hierarchical (Foundation) Data Shared with RDM

Before RMS can send purchase order, stock allocation (transfers), and work order data to RDM and accept RDM receipt, inventory adjustment, and stock count data, RDM must be set up with the necessary hierarchical (or foundation) data information. Therefore, RMS transfers supplier, store, and warehouse location information to RDM through two batch modules. Then, RMS uses another batch module to send information about items, including staples, fashion SKUs, and buyer and vendor packs. After RDM is set up with this hierarchical data, it is ready to work with purchase orders and allocations received from RMS.

Purchase Order, Allocation, and Receipt Data

In addition to the batch modules used to transfer hierarchical data to RDM, RMS has batch modules that send purchase order (PO), allocation, and related information to RDM and that receive transfer and allocation shipment receipts from RDM.

RDM also must receive advance shipment notifications from RMS. Finally, RMS sends RDM work orders. Work orders tell a warehouse what tasks must be carried out on merchandise shipments that it receives. For example, the warehouse may need to ticket items before shipping to stores.

Inventory Movement

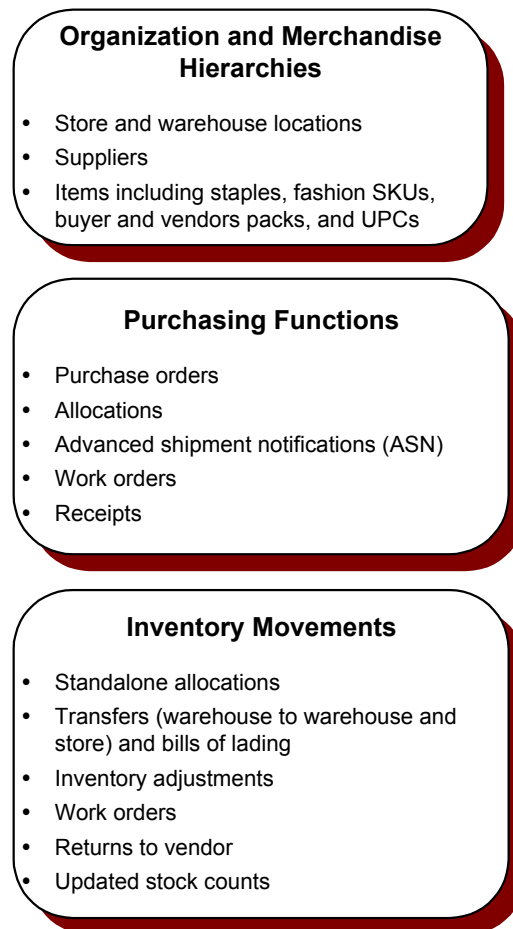
The last functional area with batch modules associated to it is inventory movement. In addition to allocating merchandise to stores based upon a purchase order, warehouses transfer items based upon stock allocations received from RMS. Other modules update RMS with inventory adjustment details, updated stock counts at the warehouse or store, and return to vendor data.

New in RMS 9.0

RMS 9.0 adds support for:

- **Tracking standalone allocations**—Standalone allocations allow you to move merchandise stored in a warehouse to other locations. RMS now tracks standalone allocations and updates RDM with that data.
- **Catch weights**—RMS no longer needs to send RDM weight values in whole numbers only. Because RDM can now accept weight values that include decimals, RMS no longer truncates the decimal, thereby sending the exact value.
- **Multiple UPCs for Items**—RMS now sends RDM the primary UPC for an item and as many additional UPCs as may be defined for that item.

The following figure summarizes a business's distribution center related information by functional areas. As described, each area has more than one batch module that processes this data between RMS and RDM.



Dataflows between RMS and RDM

RMS Batch Modules for Converting RDM Files

RMS has five additional batch modules that convert RDM data to a Retek format that can be read into various RMS tables by other RMS batch modules. These five conversion batch modules are:

- lifbolup.pc
- lifinvup.pc
- lifrtvup.pc
- lifstkup.pc
- lifrcvup.pc

Each of these five appear in the summary of RMS batch modules listed in the next section.

Summary of RMS Batch Modules in the RDM Interface

The following tables list the RMS batch modules that download data to RDM and upload data from RDM. They are grouped by the functional areas described earlier.

Note: Data transferred between the two programs must be converted to compatible file formats. Five batch modules perform these data conversion functions. Consult the *RMS 9.0 Batch Operations Guide* to see details about these and all other RMS batch modules.

RMS Batch Modules for Hierarchical Data			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
supdnld.pc	Downloads supplier data to RDM	Ad hoc Whenever supplier records are added, edited, or deleted within RMS	N/A
locdnld.pc	Downloads RMS store and warehouse location data to RDM	Ad hoc	N/A

RMS Batch Modules for Hierarchical Data			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
itemdnld.pc	Downloads merchandise item data for staples, fashion SKUs, and buyer and vendor packs to RDM	Ad hoc	N/A

RMS Batch Modules for Purchasing Functions			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
ordrev.pc	Sends approved purchase orders along with modified orders and allocation and ticketing information to RDM	Ad hoc	Run before powodld.pc and run with tsfdcdld.pc
allocupd.pc	Updates RDM with the pricing and ticketing of any previously created allocations	Ad hoc	Run after ordrev.pc
asndnld.pc	Sends RDM advance shipment notifications (ASNs)	Daily	Run after ordrev.pc

RMS Batch Modules for Purchasing Functions			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
powodld.pc	Notifies RDM of work orders to be completed on merchandise received at a location	Ad hoc	Run after ordrev.pc
rcvupld.pc	Updates RMS with shipment receipt transactions in RDM	Phase 2 (Daily) May run multiple times daily	Run after tsfparse.pc

RMS Batch Modules for Inventory Movement Data			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
tsfdnld.pc	Downloads approved warehouse transfers to RDM	N/A	Run after tsfcomb.pc
tsfiupld.pc	Uploads transfer shipment receipts from RDM	Phase 2 (Daily)	Run after tsfparse.pc
tsfdcdld.pc	Downloads incoming transfers to be received at the warehouse	Phase 4	Run with ordrev.pc and tsfdnld.pc
tsfoupld.pc	Notifies RMS of transfer shipments from warehouse to another location (store or warehouse)	Phase 2 (Daily)	Run after lifbolup.pc
tsfparse.pc	Notifies RMS of receipt information for purchase orders and transfers and receipts processed for carton information Receipts are split out for further processing by rcvupld.pc, tsfiupld.pc, and ctniupld.pc	N/A	Run after lifrcvup.pc and run before rcvupld.pc, tsfiupld.pc, and ctniupld.pc

RMS Batch Modules for Inventory Movement Data			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
invaupld.pc	Updates RMS with adjustments to total stock on hand or to unavailable inventory	Daily	Run after lifinvup.pc and after lifbolup.pc
rtvupld.pc	Updates RMS with merchandise returns to vendor(s)	Phase 2 (Daily)	Run after lifrtvup.pc and after lifinvup.pc
stkupld.pc	Updates RMS with RDM actual stock counts	Phase 3 (Daily)	Run after lifstkup.pc

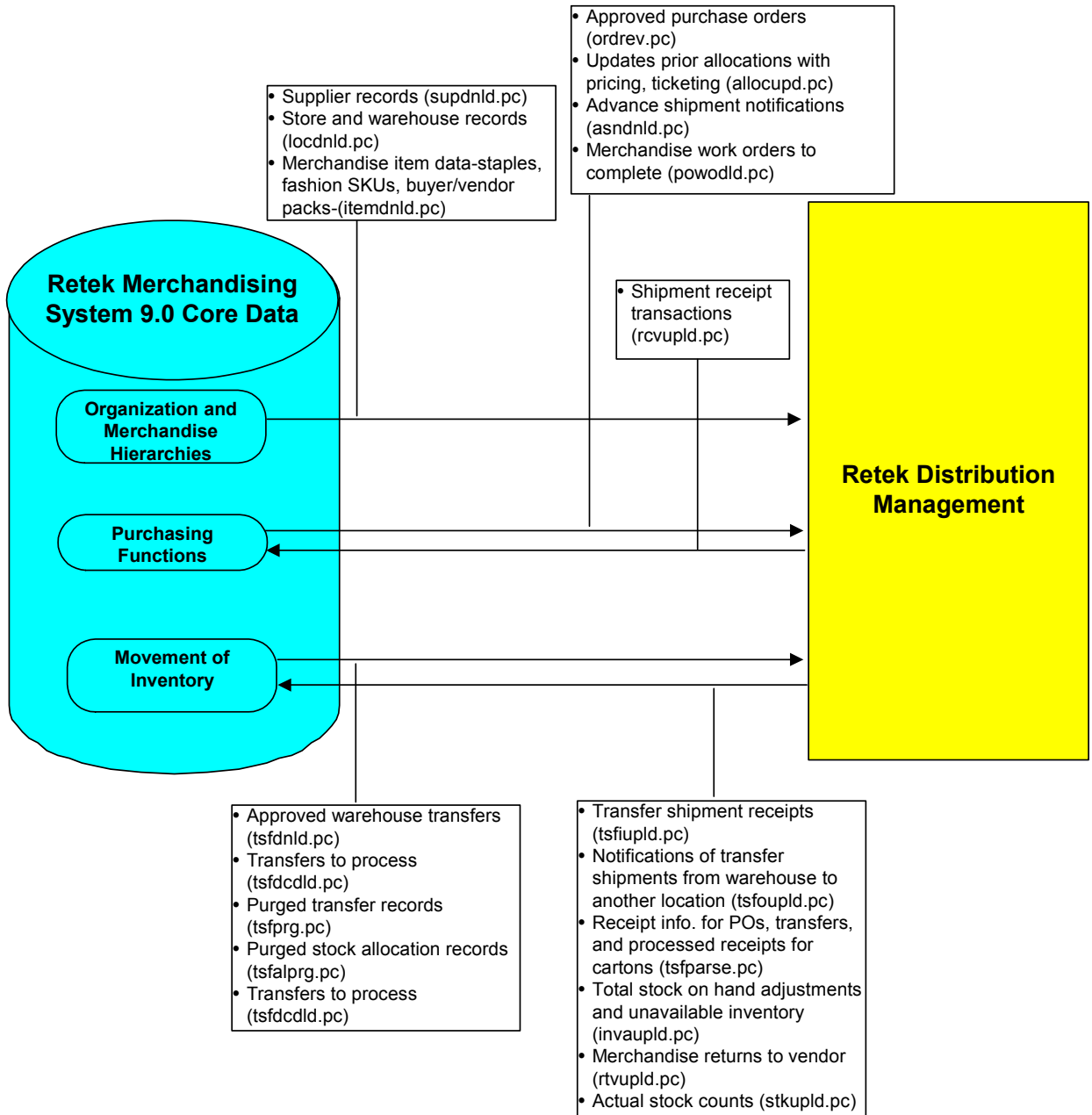
RMS Batch Modules for Purged Data			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
tsfprg.pc	Downloads purged transfer records to RDM	Ad hoc	N/A
tsfalprg.pc	Downloads purged stock allocation (transfer) records to RDM	Ad hoc	Run after tsfprg.pc
New for RMS 9.0: saaldnld.pc	Downloads standalone allocations to RDM	Ad hoc	N/A

RETEK ENTERPRISE 9.0

Dataflow Overview:

Retek Merchandising System and Retek Distribution Management

Arrows indicate flow of data. RMS batch modules listed in parentheses.



Chapter 8 - The RMS 9.0–RDW 9.0 Interface

Retek Merchandising System (RMS) and Retek Data Warehouse (RDW) batch processing work to efficiently transfer data from RMS to RDW. Because of the limited time each day that is available to RMS for running its batch schedule, and because the amount of data is frequently large, RMS batch modules use sophisticated database techniques to speed the necessary transfer of data. This document describes these techniques, the kinds of data that are updated to RDW, and the batch modules used in processing that data from RMS to RDW.

Techniques Used in Data Transfer

Retek incorporates techniques that increase the speed and efficiency of data transfers between two interfaces, such as from RMS to RDW. Two of these techniques are database triggers and SQL Loader. A description of dimension data and fact data follow in the next section.

Triggers for Dimension Data

RDW triggers are associated with RMS tables and run whenever predefined events take place such as an insertion to the table, a deletion, or an update. In RMS, triggers are used for dimension data because dimension data tables change less frequently than tables that contain data like sales transactions.

Here's an example of how a trigger is used.

Whenever a change is made to RMS's WIN_SKUS table—such as the insertion of a new SKU, a change made to an existing one, or the deletion of a SKU—any of these events cause, or “trigger”, that data to be written to a modification (MOD) table. Later, when the RDW batch schedule runs, data on the MOD table is extracted by a RDW batch module into a RDW table(s).

See the RMS Table Triggers and RDW Extraction Batch Modules section later in this document for more information.

Oracle SQL*Load Tool Used for Fact Data

The Oracle SQL*Load tool is a utility program that writes flat file data extracted from RMS to RDW. SQL*Load is highly useful in situations in which you want to mass load data into the database, thereby improving database performance by minimizing processing.

Here is an example of what happens in a SQL*Load process.

The RDW batch program SLSINVAT.PC extracts sales transaction data, promotion discount data, and inventory data from RMS. Next, it writes that data to individual flat files, also called text files. SLSINVAT.PC then starts SQL*Load. SQL*Load places the data from the flat files into RDW, based on instructions it receives from a control file.

Data in RDW and RMS: Defining Terms

RMS and RDW use different terms to talk about the kinds of data that build the structures of your database. RMS describes hierarchies for merchandise and organization. The merchandise hierarchy contains departments, classes, subclasses, styles, SKUs, and more. The organization hierarchy describes areas, regions, districts, stores, warehouses, and so on. These hierarchies are ways to classify data into meaningful groupings. Hierarchical data is also called foundation data.

RDW uses terms that are standard in the data warehousing field. Product dimension is similar to RMS's merchandise hierarchy, and organization dimension corresponds to RMS's organization hierarchy. RDW collects dimension data (information about the product and organization structures) from RMS and uses it for reference purposes. In addition to dimension data, RDW collects what it calls "fact" data from RMS batch modules.

Unlike hierarchical or dimension data, fact data come from sales transactions, pricing information, vendor shipment records, contracts, costs, and many other sources. Fact data represent transactions, such as the number of sales transactions for an item, or changes in pricing. In RDW, you typically aggregate facts. Aggregation is a process that attaches greater meaning to data by manipulating it for various purposes, such as for a report.

The following graphic illustrates some of the kinds of data that flow from RMS to RDW.

Sales and Financials

- Updated stock ledger (sales transaction) data to subclass by location by week
- Forecasted store sales and warehouse issues (via RDF)
- Uploaded transaction data from the point of sale

Allocation

- Updated supplier cost changes
- Supplier compliance and availability
- Orders built in the automated reorder cycle

Retek Sales Audit (ReSA)

- Loss prevention and sales transactions
- Transaction tender (form of payment)
- Customer-defined totals
- Voucher movement, escheated vouchers

Inventory Management

- Uploaded transfer shipments and receipts
- Uploaded receipts for merchandise and cartons
- Unavailable inventory
- Processed return-to-vendor transactions

Pricing Management

- Promotion prices changed to retail price
- Updated permanent price changes
- Updated clearance prices
- Price changes returned to regular retail price from clearance price

System Management and Core Data

- Purged records marked for deletion
- Reclassified marked items (department, class, subclass)

Samples of Data that Flow to RDW, Grouped by Functional Area

RMS Batch Modules

The batch modules listed in the following table represent those that impact RDW, either directly or indirectly. Some of the modules need to run to populate RMS tables that contain triggers that write data to RDW tables. Other modules need to run to populate tables from which RDW batch modules extract data. Finally, some modules export output files that corresponding RDW batch modules process. See the *Retek Enterprise 9.0 Batch Operations Guide* for detailed information about each of these modules.

RMS Batch Modules				
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other RMS Modules?	Run Before RDW Batch Module
<i>Foundation Data Updates:</i>				
dlyprg.pc	Purges scheduled deletions.	Daily (Phase 0)	Run first to avoid processing deleted entities.	
reclsdly.pc	Reclassifies items marked for reclassification	Daily (Phase 4)	Run after all modules run that update inventory and retail and cost prices	rclsmtxbld.pc
storeadd.pc	Adds new stores.	Daily, or as needed	N/A	
<i>Sales Transactions and Store, Warehouse Inventory Stock Updates:</i>				
posupld.pc	Writes sales transaction data to the RMS table TRAN_DATA. Another RMS batch module, salstage.pc, then writes TRAN_DATA to a mirrored table called IF_TRAN_DATA from which the RDW batch module slsinvat.pc then extracts data for updating its own transaction tables. For the Enterprise 9.0 release, posupld.pc also writes pack sales and returns to TRAN_DATA.	Daily (Phase 3)	Run before salstage.pc	slsinvat.pc

RMS Batch Modules				
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other RMS Modules?	Run Before RDW Batch Module
tsfoupld.pc	Writes information about inventory that has been shipped.	Daily (Phase 2)	Varies. See <i>Operations Guide</i> .	
tsfiupld.pc	Writes information about inventory that has been received.	Daily (Phase 2)	Varies. See <i>Operations Guide</i> .	
salstage.pc	Writes sales transaction data from TRAN_DATA to IF_TRAN_DATA	Daily	Run after posupld.pc	slsinvat. pc ivusldat. pc inasldat. pc ivmsldat. .pc
<i>Shipment Receipt, Transfer, and Purchase Order Updates:</i>				
ctniupld.pc	Writes information about cartons received as a result of a purchase order or transfer.	Weekly (Phase 3)	See the Scheduling Constraints section for this module in the <i>Operations Guide</i> .	semildat. .pc scrclat. pc scrqlat. pc setusdat. pc
rcvupld.pc	Writes transactions about items received from a purchase order.	Daily (Phase 2)	See the Scheduling Constraints section for this module in the <i>Operations Guide</i> .	semildat. .pc scrclat. pc scrqlat. pc setusdat. pc
<i>Replenishment, Supplier Availability, and Return-to-Vendor Updates:</i>				
ediupavl.pc	Writes EDI notifications from vendors of availability schedules.	Daily (Phase 1)	N/A	savusdat. .pc

RMS Batch Modules				
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other RMS Modules?	Run Before RDW Batch Module
cntrprss.pc	Writes evaluation of contracts available to be used for replenishing stock.	Run on the same schedule as the replenishment cycle.	Runs before rplblds.pc and after rplexts.pc	sctusdat.pc
rplbld.pc	Builds orders	Daily (Phase 3) in the replenishment cycle.	Run after rplexts.pc and cntrprss.pc	
rplprg.pc	Evaluates automatically created orders to determine whether they meet minimum vendor requirements. If not, they are purged.	Phase 3.	Run after rplbld.pc	
rtvupld.pc	Updates return-to-vendor transactions.	Daily (Phase 2)	See the Scheduling Constraints section for this module in the <i>Operations Guide</i> .	ivrsldat.pc
<i>Pricing Updates:</i>				
prmext.pc	Transfers changes in promotional prices.	Daily (Phase 1)	N/A	prcsldat.pc
pcext.pc	Transfers pending regular price changes.	Daily (Phase 3)	After pccext.pc	prcsldat.pc
pccext.pc	Transfers clearance price changes.	Daily (Phase 3)	N/A	prcsldat.pc
pccrext.pc	Transfers resets of clearance prices back to regular retail prices.	Daily (Phase 3)	N/A	prcsldat.pc
<i>Supplier Cost, Weekly Sales Rollup, and Forecasted Sales Updates:</i>				
sccext.pc	Writes supplier cost changes.	Ad hoc.	N/A	cstslat.pc

RMS Batch Modules				
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other RMS Modules?	Run Before RDW Batch Module
salweek.pc	Writes rolled-up weekly stock ledger (transaction) data by subclass and location.	Run with stock ledger, anytime after end of week.	N/A	stlblwat.pc
fcslupld.pc	Writes sales forecasts received from Retek Demand Forecasting.	Daily.	N/A	sfeslwat.pc
<i>Retek Sales Audit</i>				
saexprdw.pc	Creates RDWT output file for sales transactions and loss prevention	N/A	N/A	stslldmat.pc
saexprdw.pc	Creates RDWF output file for transaction tenders	N/A	N/A	ttldmat.pc
saexprdw.pc	Creates RDWC output file for loss prevention totals (drawer over or short)	N/A	N/A	lptotcldat.pc
saexprdw.pc	Creates RDWS output file for user-defined totals	N/A	N/A	lptotldat.pc

RDW's Data Warehouse Interface Batch Modules

RDW batch modules that run within RMS's batch schedule are known as the Data Warehouse Interface (DWI). These RDW batch modules extract fact data from RMS tables and write that data to individual flat files (also called text, or ASCII files). Using Oracle's SQL*Load process, the data in the flat files are then populated to RDW staging tables that, in turn, are used by additional batch modules within RDW to populate and then aggregate RDW tables.

This section describes the DWI batch modules, the kind of data extracted from RMS tables, the RMS table names from which data are extracted, and the RDW tables that are finally populated with the data.

RDW DWI Batch Modules		
RDW Batch Module (functional area)	Description	Timing and Dependencies
cmptrprcsdat.pc (competitor pricing)	Extracts data from the RMS competitor price history table COMP_PRICE_HIST and the RMS link table COMP_STORE_LINK to populate RDW's CMPTR_PRICING_SKU_LD_AT table	N/A
cstslldat.pc (cost)	Extracts cost data from PRICE_HIST and PERIOD and writes to RDW's COST_SKU_LD_AT	Run after RMS's sccext.pc and before RDW's cstslldm.pc
exchngrateat.pc (exchange rates)	Extracts exchange (currency) rates from RMS's CURRENCY_RATES and writes the records to a flat file for SQL*Loading to RDW's EXCHNG_RATE_CRNCY_DAY_AT	Run daily, before RDW's exchngrateatdm.pc
ivaslldat.pc (inventory adjustments)	Extracts RMS total stock-on-hand adjustments from IF_TRAN_DATA to populate RDW's INV_ADJ_SKU_LD_AT	Run daily, after RMS's salstage.pc and before RDW's ivaslldm.pc
ivmslldat.pc (inventory movement)	Extracts RMS inventory movement data from RMS's IF_TRAN_DATA to populate RDW's INV_MOVE_SKU_LD_AT	Run daily, after RMS's salstage.pc and before RDW's ivmslldm.pc
ivrsldat.pc (return to vendor)	Extracts RMS return-to vendor inventory facts from RTV_HEAD and RTV_DETAIL to populate RDW's INV_RTV_VEND_SLD_AT	Run daily, after RMS's rtvupld.pc and before RDW's ivrsldm.pc
ivuslldat.pc (unavailable inventory)	Extracts RMS non-salable, unavailable inventory facts from IF_TRAN_DATA to populate RDW's INV_UNAVL_SKU_LD_AT	Run daily, after RMS's salstage.pc and before RDW's ivuslldm.pc

RDW DWI Batch Modules		
RDW Batch Module (functional area)	Description	Timing and Dependencies
lptotclat.pc	If the customer uses ReSA:	Run daily, after ReSA's saexprdw.pc and before RDW's lptotclddm.pc
lptotldat.pc (user-defined totals, ReSA)	If the customer uses ReSA: Processes the RDWS (store/day user-defined totals) output file produced by ReSA's saexprdw.pc by SQL*Loading an output file to RDW's LP_TOT_LD_AT	Run daily, after ReSA's saexprdw.pc and before RDW's lptotlddm.pc
nctusldat.pc (net cost and profit)	Extracts data from RMS's DEAL_SKU_COST table and writes them to RDW's NET_COST_SUPP_SLD_AT table	Run daily after RMS's dealcalc.pc and before RDW's nctusliddm.pc
prcsldat.pc (pricing)	Extracts pricing information from RMS tables PRICE_HIST, ITEM_ZONE_PRICE and PERIOD and writes data to RDW's PRICING_SKU_LD_AT	Run after RMS's prmext.pc, pccext.pc, pccext.pc, and pccrxt.pc and before RDW's prcslddm.pc
rsndm_pre.pc (reason)	Extracts data from RMS's CODE_DETAIL table	Run before RDW's rsndm.pc
savusdat.pc (supplier availability)	Extracts supplier availability for a particular SKU (for contract types A and D) from RMS's SUP_AVAIL table to RDW's VCMP_AVAIL_VNDR_SKU_DAY_A T	Run after RMS's rplprg.pc and ediupavl.pc
scmidldat.pc (supplier compliance)	Extracts supplier compliance data for missed deliveries from RMS tables: SOURCE_DLVRY_SCHED SOURCE_DLVRY_SCHED_DAYS SHIPMENT and writes them to RDW's SCMP_RCPT_MISS_LD_AT table	Run after RMS's rcvupld.pc (receiving upload) and ctniupld.pc (carton receiving upload) Run before RDW's semisldat.pc and scmioldat.pc

RDW DWI Batch Modules		
RDW Batch Module (functional area)	Description	Timing and Dependencies
scmioldat.pc	Extracts supplier compliance data about shipments with advance notice that were not received from RMS tables:	Run after RMS's rcvupld.pc (receiving upload) and ctniupld.pc (carton receiving upload)
scmisldat.pc (supplier compliance)	Extracts supplier compliance data for missed shipments (those not received by the estimated arrival date) from RMS tables: SHIPMENT ORDHEAD and writes them to RDW's SCMP_RCPT_MISS_LD_AT table	Run after RMS's rcvupld.pc (receiving upload) and ctniupld.pc (carton receiving upload) Run after RDW's scmidldat.pc and before RDW's scmioldat.pc
scqcat.pc (supplier compliance)	Extracts RMS data for supplier compliance quality control from the SHIPMENT and QC table and writes them to RDW's SCMP_QC_SKU_LD_AT table	Run before RDW's scqcdm.pc
scrclldat.pc (supplier compliance)	Extracts supplier compliance data for timeliness of delivery from RMS tables: SCMP_SHIP_MD SHIPMENT ORDHEAD and writes them to RDW's SCMP_RCPT_LD_AT	Run before RDW's scrclddm.pc
scrqtlldat.pc (supplier compliance)	Extracts supplier compliance data for quantity from RMS tables: SCMP_SHIP_SKU_MD SHIPMENT ORDHEAD SHIPSKU ORDLOC and writes them to the RDW table SCMP_RCPT_QTY_LD_AT	Run before RDW's scrqtlldm.pc

RDW DWI Batch Modules		
RDW Batch Module (functional area)	Description	Timing and Dependencies
sctusdat.pc	Extracts vendor compliance contract data from these RMS tables:	Run after RMS's ctniupld.pc, cntrprss.pc, and rcvupld.pc and before RDW's vctsddm.pc
sfclwat.pc (forecasts for store sales and warehouse issues via RDF and RMS)	Extracts forecasted sales and warehouse issues from RMS' ITEM_FORECAST table and writes to RDW's SLS_FCST_SKU_LW_AT table.	Run after RMS's fcslupld.pc and before RDW's sfclwdm.pc
slsinvat.pc (stock ledger merchandise sales, promotional sales, inventory position)	Extracts data from RMS for: Stock ledger financial transaction data from IF_TRAN_DATA to RDW's SLS_SKU_LD_AT SKU/store promotional discount sales from DAILY_SALES_DISCOUNT to RDW's SLS_SKU_LD_PRMTN_SCHM_A T Inventory—(snapshot of stock on hand for sales and returns at end of day for staple and fashion SKUs) from WIN_STORE, WIN_WH, RAG_SKUS_ST, and RAG_SKUS_WH to RDW's INV_SKU_LD_AT	Run after RMS's posupld.pc and salstage.pc and before RDW's invslddm.pc, slsslddm.pc and prsslddm.pc
stlblwat.pc (stock ledger)	Extracts weekly sales transactions (rolled up to subclass/location/week) from RMS's WEEK_DATA to RDW's INV_VAL_SBC_LW_AT	Run after RMS's salweek.pc and before RDW's stlblwdm.pc

RDW DWI Batch Modules		
RDW Batch Module (functional area)	Description	Timing and Dependencies
stslmat.pc (loss prevention, sales transactions, ReSA)	If the customer uses Retek Sales Audit (ReSA), processes the RDWT (point-of-sale transaction data) output file (one file for each location) produced by ReSA's saexprdw.pc for: loss prevention: it writes to RDW's LP_TRAN_LM_AT sales and returns: it writes to RDW's SLS_TRAN_SKU_LM_AT	Run daily, after ReSA's saexprdw.pc and RMS's posupld.pc and before stslmdm.pc (for SLS_TRAN_SKU_LM_AT) and lptldm.pc (for LP_TRAN_LM_AT)
ttldmat.pc (transaction tender, ReSA)	If the customer uses ReSA: Processes the RDWF (transaction tender/form-of-payment) output file produced by ReSA's saexprdw.pc by SQL*Loading an output file to RDW's TNRD_TRAN_LM_AT	Run daily, after ReSA's saexprdw.pc and before RDW's rgstrdm.pc and ttldm.pc
vchreschdat.pc (escheated vouchers, ReSA)		
vchrmovaldat.pc (voucher movement, ReSA)		
vchroutlwat.pc (voucher movement, ReSA)	Extracts data from ReSA's SA_VOUCHER and writes them to RDW's VCHR_OUT_LW_AT table	Run after ReSA's savouch.pc

RMS Table Triggers and RDW Extraction Batch Modules:

As described earlier, triggers reside within various RMS tables for the purpose of causing any data added to or modified on that table to be written to a RDW table from which a RDW batch module later extracts the data for inclusion on a RDW table(s). The following table shows four examples of RMS hierarchical data that enters RDW dimension tables as a result of a trigger mechanism built into the RMS table. Both the modification table (called a "MD" table in RDW) where the data is written and the respective RDW extraction module are listed in the table.

Consult the Retek Data Warehouse 9.0 Operations Guide for detailed information about all RDW table triggers.

Sample Table Triggers			
<i>Note: This list is only representative of table triggers. See the RDW 9.0 Operations Guide for a complete list.</i>			
<i>RMS Table</i>	<i>Trigger Name in RMS Table</i>	<i>RDW Table (to which triggered data is written)</i>	<i>RDW Extraction Program</i>
Fashion SKU: Modifications to RMS's RAG_SKUS table cause the trigger to write to the RDW table	RDW_RAG_SKUS_TRIGGER	PROD_SKU_MD	prdskuat.pc
Subclass: Any modification to RMS's SUBCLASS table will trigger a modification to the RDW table	RDW_SUBCLASS_TRIGGER	PROD_SBCLASS_MD	prdsbcac.pc
Staple SKU: A modification to RMS's WIN_SKUS causes a modification to be written to the RDW table	RDW_WIN_SKUS_TRIGGER	PROD_SKU_MD	prdskuat.pc
Store: Modifications to various RMS tables, beginning with STORE, and including PERIOD, STORE_ATTRIBUTES, TSFZONE cause the trigger to write to the RDW table.	RDW_STORE_TRIGGER	ORG_LOC_MD	orglocat.pc
Vouchers: ReSA "moved" vouchers (those that are sold or redeemed) from ReSA's SA_VOUCHER table to the RDW table	RDW_VOUCHER_TRIGGER	VCHR_MOVE_LD_MD (for 'move' vouchers), or VCHR_ESCH_DAY_MD (for 'escheat' vouchers)	vchrmoveldat.pc (for 'move' vouchers), or vchreschdat.pc (for 'escheat' vouchers)

Chapter 9 - Point of Sale Data Upload to RMS

The Retek Merchandising System (RMS) includes a convenient interface to your point-of-sale system (POS) that allows you to upload sales transaction data quickly. RMS is able to accomplish these POS uploads because of the efficiency of its batch module that accepts your *rolled-up* and formatted sales transaction data files into RMS. Once the data enters RMS, other modules take over the posting of that data to daily and weekly historical tables. This document describes the upload and validation of sales transaction data from your POS to RMS using a batch module called POSUPLD.PC and other modules used in the batch processing of that data for posting to RMS tables.

The POS Upload Process

Before RMS can accept sales transaction data, you need to ensure that your data is correctly prepared. Then it is uploaded.

Preparing Transaction Data for the Upload

Two tasks need to be accomplished before transaction data is ready for upload to RMS. Initially you roll up data in your transaction logs (called TLOGs), and then you convert the rolled-up files into a format that RMS can use.

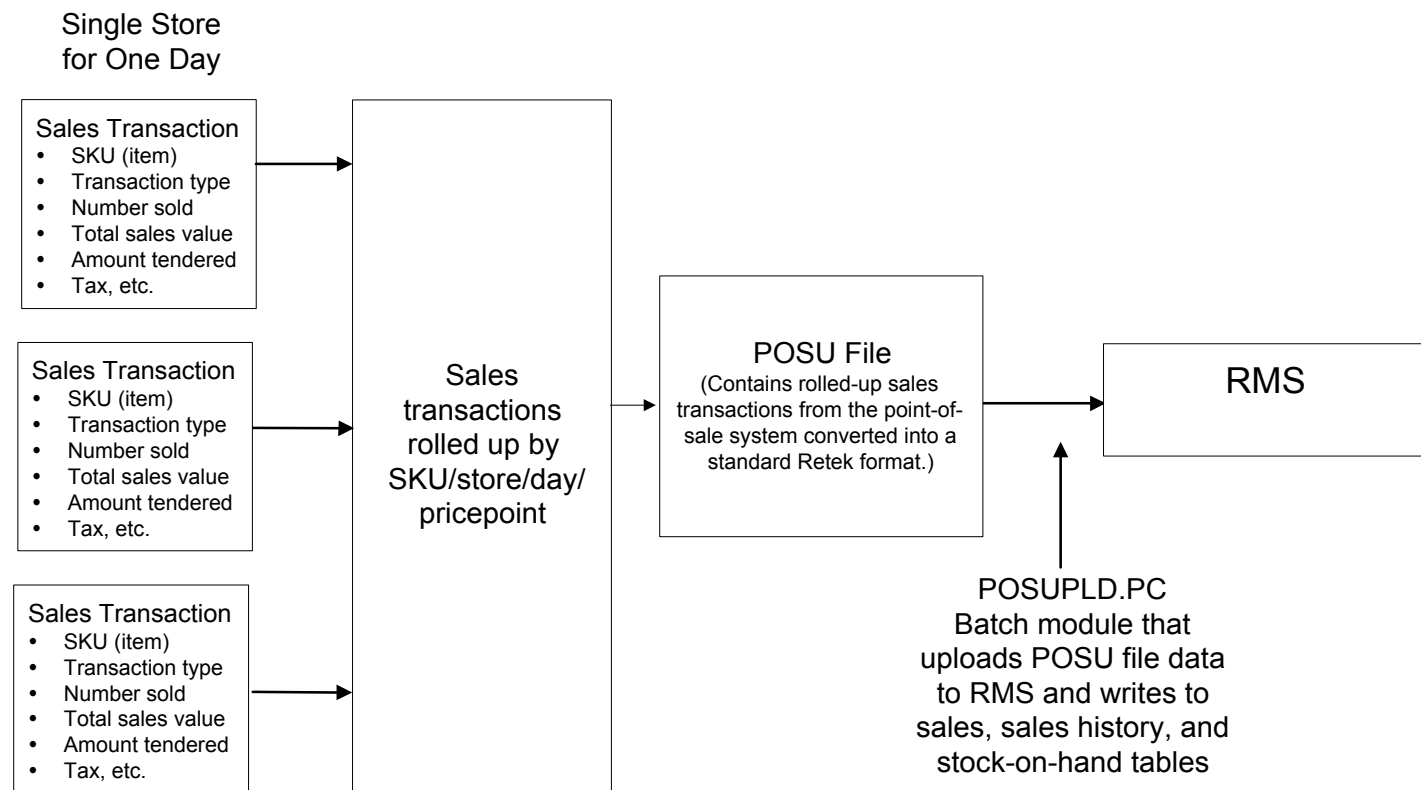
Because you record transactions by the item sold, price, tax on the sale, amount tendered, and so on, you need to roll up these records to the SKU/day/store/price point level. Recording transactions by SKUs (stock keeping units) is fundamental to RMS because a SKU provides the most exact definition of an item within the database. The result of the rollup is that your transactions are described as the number of each item sold at a particular price at a store on one day.

After you roll up transaction data, the second task is to convert it to a format that RMS can read. The result of these two tasks is that each store has what's called a POSU file, a file that is ready for upload to RMS. At this point, you are ready to run the batch module POSUPLD.PC.

Uploading Transaction Data

POSUPLD.PC is RMS's batch module that uploads your sales transaction data into RMS and then posts that data to the appropriate sales transaction, sales history, and stock-on-hand tables. The following graphic illustrates the upload process.

POS Upload Process



How Sales Transaction Data is Uploaded into RMS

Note: The transaction type for a sale (as opposed to a return) can be a sale at a regular price, promotional price, or clearance price. Each sale price is considered a “pricepoint”. Sales transactions are then rolled up to the SKU/store/day/pricepoint. The batch module POSUPLD.PC may run several times a day for each store location.

Validating Total Amounts

Rolled-up sales transactions for individual items at the store are validated within POSUPLD.PC. Before looking at how the validation is performed, take a closer look at a list of sales transactions and how they are rolled up. Suppose that a store sells an item that is identified as SKU number 1234. During the day, sales for SKU 1234 might look like this:

Sales for SKU Number 1234 (at one store during one day)			
Transaction Number	Number of Items Sold	Amount (in specified currency unit)	Pricepoint (price reason)
167	1	9.99	Regular
395	2	18.00	Promotional
843	1	7.99	Clearance
987	3	27.00	Promotional
1041	1	9.99	Regular
1265	4	31.96	Clearance

Note: The variation of the price per item in different transactions. This results from the price applied at the time of sale, the pricepoint. Now look at the next table that shows the same transactions rolled up by SKU and pricepoint.

Number of Items Sold	Price Reason (pricepoint)	Total Amount for SKU/Pricepoint (in currency)
2	Regular price	19.98
5	Promotional price	45.00
5	Clearance price	39.95

The Validation Process

After sales transactions are rolled up, POSUPLD.PC then takes the totals and looks for any discounts for transactions in the POSU file. It applies the discounts to an expected total dollar amount using the price listed for that SKU from the pricing tables. It next compares this expected total against the reported total. If the program finds a discrepancy between the two amounts it is reported. If the two totals match, the rollup is considered valid. If value-added tax (VAT) is included in any sales transaction amounts, it is removed from those transactions prior to the validation process.

Posting Sales Transaction Data in RMS

Sales transactions are written to sales, sales history, and stock-on-hand tables in RMS by POSUPLD.PC.

Additional batch modules in the batch process then begin to post that data to other RMS tables. The module HSTBLD.PC writes SKU-based transactions to historical sales data tables by subclass, class, and department. HSTBLD.PC runs daily after POSUPLD.PC. Other modules update staple and fashion SKU tables weekly and purge older data in fashion SKU tables. Here is a list of modules that are involved in the process of uploading and posting transaction sales data.

Traditional Sales Posting Batch Modules			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
posupld.pc	Uploads rolled-up, formatted sales transaction data from each store location and posts them to sales, sales history, and stock-on-hand tables within RMS.	Daily (Phase 2) Usually runs multiple times daily for each store.	Run at the beginning of the batch run during the POS polling cycle.
hstbld.pc	Writes sales transaction data to history tables for subclass (SUBCLASS_SALES_HIST), class (CLASS_SALES_HIST), and department (DEPT_SALES_HIST).	Daily (Phase 3) Weekly (Ad Hoc)	Run after posupld.pc Run before prepost.pc with the argument hstbld_post
prepost.pc (with the argument hstbld_post)	Generic program used in this process to purge temporary tables built during the roll-up. This batch contains a function you specify that purges these tables. Prepost is used with other batch modules for table deletions and mass updates.	Daily, or as needed after running hstbld.pc.	N/A
hstwksst.pc	Weekly update of the WIN_STORE_HIST table with current stock on hand and retail value for staple SKUs at the store.	Weekly (Phase 3)	N/A
hstwkswh.pc	Weekly update of the WIN_WH_HIST table with current stock on hand and retail value for staple SKUs at the warehouse.	Weekly (Phase 3)	N/A

Traditional Sales Posting Batch Modules			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
hstwkfst.pc	Weekly update of the RAG_SKUS_ST_HIST table with current stock on hand and retail and average cost values for fashion SKUs at the store.	Weekly (Phase 3)	N/A
hstwkfwh.pc	Weekly update of the RAG_SKUS_WH_HIST table with current stock on hand and retail and average cost values for fashion SKUs at the warehouse.	Weekly (Phase 3)	N/A
hstprg.pc	Purges older data in the fashion store and warehouse history tables RAG_SKUS_ST_HIST and RAG_SKUS_WH_HIST.	Ad hoc (monthly).	N/A

A Special Note about Sales Audit 9.0 and POSUPLD.PC

Retek Enterprise 9.0 offers customers an optional program called Retek Sales Audit (ReSA). Unlike the standard POS upload process to RMS that is described in this document, ReSA accepts POS data at the transaction level for the store/day. The standard POS upload process described earlier requires the customer to *roll up* individual transactions to the SKU/store/day/pricepoint level. ReSA validates individual sales transactions for a store day, offers a method to build and apply business audit rules, and lets users reconcile transaction errors, all prior to creating an output file rolled up to the SKU/store/day/pricepoint level that POSUPLD.PC batch processes to RMS.

Chapter 10 - Updating the POS System from RMS

Retek Merchandising System (RMS) updates a customer's point-of-sale (POS) system with pricing and related data through a batch module called POSDNLD.PC. The module updates the POS system with data contained in a RMS table called POS_MODS. POS_MODS, in turn, receives data from eight other RMS batch modules. This document describes the kinds of data that these eight modules insert into the POS_MODS table and how POSDNLD.PC transfers that data to the POS system.

Note: Pos_mods records are also created in the POS_MODS table from SKU-related transactions that users create online in RMS. For example, you can create a SKU, associate a SKU with a store, or associate a UPC with a SKU...all online. However, this document focuses on the data that batch processing modules insert into the pos_mods table.

The POS Download Process

The POS download process begins with the eight batch modules that write price changes and related information to the POS_MODS table. The eight batch modules and the kind of data that the eight write to POS_MODS are:

PCCDNLD.PC: Clearance Pricing POS Download. Writes clearance markdowns to POS_MODS that are to take effect within a predetermined number of days.

PCDNLD.PC: Price Change POS Extract. Writes permanent price change records that are scheduled to take effect in a pre-determined number of days to the POS_MODS table.

PCCRDNLD.PC: Clearance Reset Pricing POS Extract. Writes price resets to the POS_MODS table. A reset price is one that existed prior to a clearance price period. The reset prices are scheduled to take effect in a pre-determined number of days.

PRMEXT.PC: The Promotion Price Extract and Download. Writes temporary price markdowns for a set period for individual SKUs in a store.

PCTRANDN.PC: Price Transactions Download for Store Zone Changes. Writes price changes for a particular store that has been rezoned.

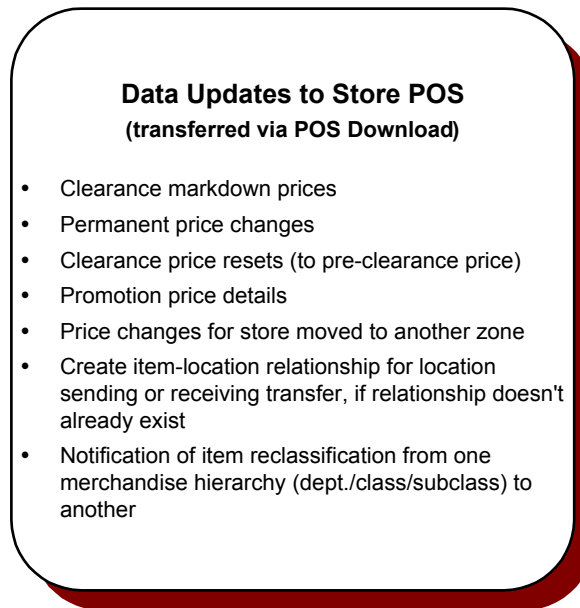
TSFOUPLD.PC: Transfer Shipments Upload. Creates the item/location relationship for the location sending the transfer if it doesn't already exist.

TSFIUPLD.PC: Transfer Receipts Upload. Creates the item/location relationship for the location receiving or sending the transfer, if it doesn't already exist.

Note: Other batch modules, in addition to tsfoupld.pc and tsfiupld.pc, that may create a new item/location relationship include posupld.pc, ediupack.pc, stkupld.pc, and rplbld.pc. In all of these batch modules, the creation of the item/location relationship is only a secondary function that each performs as necessary.

RECLSDLY.PC: Item Reclassification. Notifies a store that an item is reclassified to another merchandise hierarchy (department/class/subclass).

This graphic summarizes the kinds of data that are transferred to the POS system from the POS_MODS table and POSDNLD.PC.



Converting, Downloading, and Clearing the POS_MODS Table

After the eight batch modules finish writing pricing data changes to the POS_MODS table, POSDNLD.PC copies the data to a flat, or text, file. A flat file is created because the customer can easily convert this standard file format to the format used in its POS system. POSDNLD.PC transfers this flat file to the customer's POS whenever the module runs in the batch schedule.

After POSDNLD.PC transfers the flat file to the POS, another batch module, called PREPOST.PC, runs to clear the POS_MODS table. The table must be cleared before the eight batch modules can write the next day's pricing updates. PREPOST.PC is a unique batch module. It carries out a number of functions before or after other batch modules run. POSDNLD.PC calls one of PREPOST.PC's functions that deletes all data in POS_MODS.

Batch Modules in the POS Download Process

This table lists all batch modules that are involved in the process of updating the POS system.

POS Download Batch Modules			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
pccdnlld.pc	Writes clearance markdowns to the POS_MODS table that are to take effect within a predetermined number of days.	Daily (Phase 1)	Run before pccrdnlld.pc within the clearance batch component.
pcdnld.pc	Writes permanent price change records to the POS_MODS table that are scheduled to take effect in a pre-determined number of days.	Daily (Phase 1)	Run during the regular pricing batch component.
pccrdnlld.pc	Writes price resets to the POS_MODS table. A reset price is one that existed prior to a clearance price period. The reset prices are scheduled to take effect in a pre-determined number of days.	Daily (Phase 1)	Run after pccdnld.pc in the clearance batch component.
prmext.pc	Writes promotion price details to the POS_MODS table.	Daily (Phase 1)	Run after prmxpld.pc within the promotions batch component.
pctrandn.pc	Writes price changes for a particular store that has been rezoned to the POS_MODS table.	Daily (Phase 3)	Run after pctranex.pc within the regular pricing batch component.
tsfoupld.pc	Creation of a new item/location relationship for the location receiving the transfer. Writes to the POS_MODS table.	Phase 2	Run before tsfiupld.pc within the transfer batch component.

POS Download Batch Modules			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
tsfiupld.pc	Creation of a new item/location relationship for the location that either receives or sends the transfer. Writes to the POS_MODS table.	Phase 2	Run after tsfoupld.pc and tsfpars.pc in the transfer batch component.
Note that like tsfoupld.pc and tsfiupld.pc, each of these batch modules may create a new item/location relationship on POS_MODS: posupld.pc, ediupack.pc, stkupld.pc, and rplbld.pc			
reclsdl.pc	Notification to a store that an item has been reclassified to another merchandise hierarchy (department/class/subclass).	Phase 4	N/A
posdnld.pc	Converts the POS_MODS table to a flat (text) file, transfers it to the POS system, and clears the POS_MODS table using the posdnld_post function of the prepost.pc module.	Ad hoc	N/A

Chapter 11 - Retek Invoice Matching Batch Modules

Invoice Matching allows you to verify your received supplier shipments (receipts) against invoices and to confirm that the goods received are within cost and quantity tolerances that are defined for that supplier. You can receive invoices from a supplier in one of two ways: via an electronic data interface (EDI) or on paper. If you receive invoices through EDI, they can automatically enter the invoice matching tables. Information from paper invoices must be entered manually.

You can match invoices to receipts either manually or through an automated batch process. Invoice matching uses a set of cross-reference numbers, including receipt numbers, purchase order numbers, locations, and/or advance shipping notice (ASN) numbers.

Employees with the proper authorization can approve specific invoice matches. After an invoice is matched and approved, it is ready to be pulled into your accounts payable system for processing.

Retek Merchandising System's (RMS) invoice matching feature offers you the ability to:

- Upload invoices and credit notes from the electronic data interchange (EDI)
- Match invoices to supplier receipts (in the SHIPMENT and SHIPSKU tables) by verifying goods received in a shipment and associated costs and then automatically approve invoices for payment
- Reconcile discrepancies found in the invoice matching process by communicating information back to the supplier through EDI
- Send invoice records marked for payment to Oracle Financial Application 10.7 Accounts Payable
- Close shipments that remain open for a specified number of days and that are not associated with open invoices
- Purge old invoices not previously purged, including those that are non-merchandise. Purging is based on the invoice post date
- Match invoices in support of direct store delivery, both merchandise and non-merchandise

Supplier records must be entered before using invoice-matching features. Typically, supplier information is interfaced into RMS through a company's financial system. RMS also captures other supplier-specific information that is maintained in RMS and used throughout the system.

Invoice Matching Process

The module that actually attempts to match invoices to receipts, `invmatch.pc`, can match invoices and receipts at two levels. First, it tries to match the summary of total cost and, optionally, total quantity on the invoice to the total cost and total quantity on the receipt. If these values do not match, the program then attempts to match on a line-by-line basis. That is, it tries to match the item cost and quantity for one line on the invoice to the item cost and quantity for the same line on the receipt.

Invoices that are unmatched remain in that status until they are completely matched to receipts or until they are resolved with a debit memo or credit note from the supplier. After matching, they receive a status of “Matched.” After they have been matched, you can approve them manually or have them automatically approved by the program.

After approval, the module `invcpst.pc` writes them to the staging tables. Once in the staging tables for Oracle Financial Application 10.7 Accounts Payable, they are considered posted and ready to pay in Accounts Payable.

Batch Modules at a Glance

This table shows you an overview of RMS batch modules that are associated with invoice matching. See the Invoice Matching Process section for further information.

Invoice Matching Batch Modules			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
ediupinv.pc	Uploads invoices and credit notes from the EDI into the invoice-matching table	Daily, or as needed	N/A
edidldeb.pc	Reconciles discrepancies found in invoice matching by sending debit memos or credit note requests back to the supplier through the EDI	Daily, or as needed	N/A
invcshp.pc	Close shipments not associated with an open invoice that have remained open for a specified number of days	As necessary	Run after invcpst.pc
invmatch.pc	Match invoices to supplier bills/receipts by confirming goods received and costs Matched invoices are automatically approved only if the supplier-level option is set for auto-approval of invoice	Run daily	Run after ediupinv.pc (invoice upload from EDI) and after rcvupld.pc Run before invcpst.pc
invcpst.pc	Move matched invoices to the RIM staging tables	Run daily	Run after ediupinv.pc Run after invmatch.pc
invprg.pc	Purge old, posted invoices (including non-merchandise invoices),	Monthly	N/A

Invoice Matching Batch Modules			
Batch Module Program Name	What It Does	When to Run It	Run Before/After Other Modules?
saexpim.pc	Provides invoicing support for Direct Store Delivery (DSD) by transferring data input into the POS (and imported by ReSA) to RIM which uses that data to create an invoice for DSD		Run after saescheat.pc
fifinvcu107.pc	Transfers data from RIM staging tables (IIF_HEAD, etc.) to Oracle Financial Application 10.7 Accounts Payable tables: AP_EXPENSE_REPORT_HEADERS_ALL AP_EXPENSE_REPORT_LINES_ALL	Daily	Run after invcpst.pc

Chapter 12 - The Retek Enterprise 9.0 - Oracle Financial Application 10.7 Interface Overview

Retek Enterprise 9.0 fully supports an interface to Oracle Financial Application 10.7's General Ledger (Oracle GL), Accounts Payable (Oracle AP), and Purchasing (Oracle PO) systems for the exchange of financial data. Retek Merchandising System (RMS) updates the Oracle GL with sales transaction data and purchase orders and receipts (for Retek customers not using Retek Invoice Matching). Those customers using Retek Invoice Matching (RIM) can send invoices to the Oracle AP that are matched and approved for payment. Retek Sales Audit (ReSA) sends the Oracle GL total values for a store's daily sales, tenders, paid-ins, paid-outs, and taxes.

For its part, the Oracle AP updates RMS with reference data such as currency rates, freight and payment terms, and suppliers and supplier addresses.

This document describes the kinds of data that are shared between the Retek Enterprise and Oracle Financial Application 10.7 and the Retek batch modules that process data to and from Oracle.

A Note about Clearing Accounts

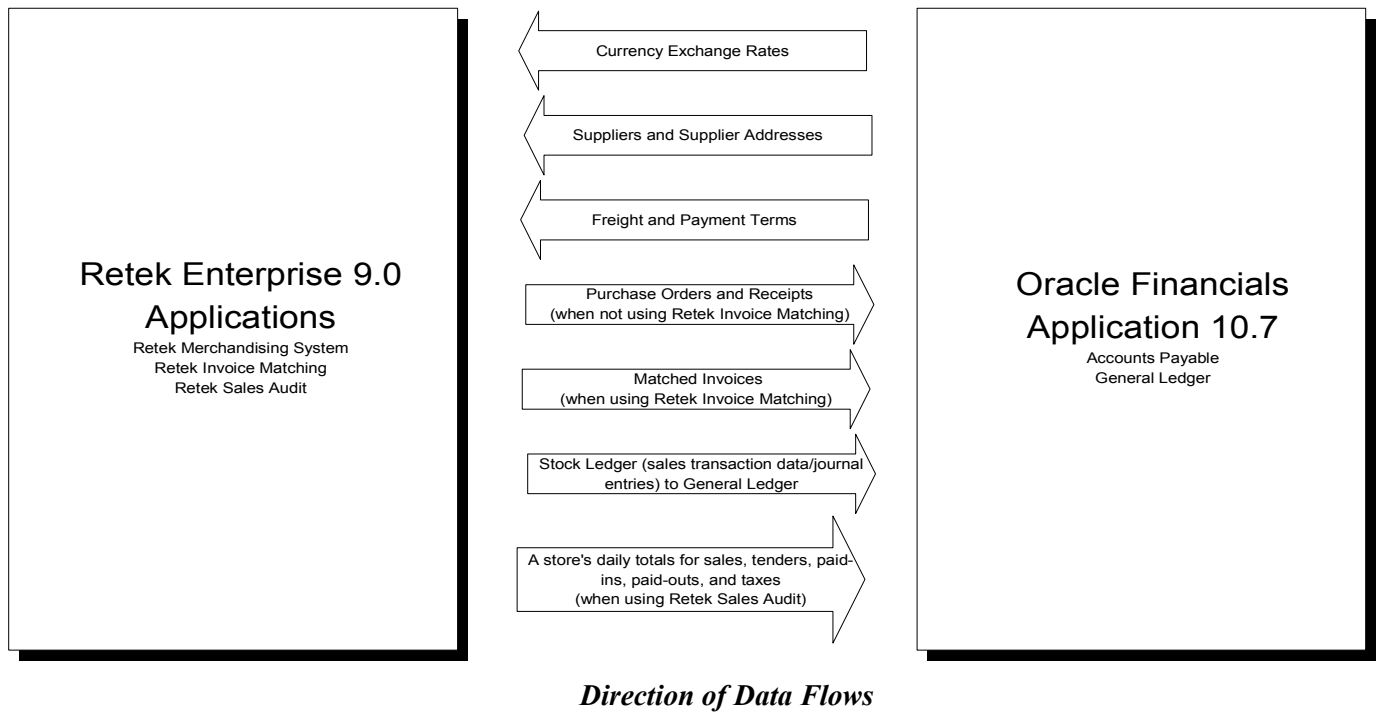
The customer is responsible for setting up clearing accounts for ReSA, RMS, and RIM when integrating to any general ledger package. The ReSA clearing account is used as the offset to the primary general ledger entry for all totals. For example, when setting up cash as a total, the primary entry is typically a debit. When setting up sales, the primary entry is typically a credit. The interface requires an offset account for every entry. The retailer could use "ReSA Clearing" as the name of such an account. At the end of the export of totals to the general ledger, ReSA Clearing would also export the offset account, the contents of which should exactly match, thus netting to zero.

Knowing the Flow: Who “Owns” the Data

Use the chart that follows to keep track of the direction that data flows between the Retek Enterprise and Oracle Financial Application 10.7. Knowing who owns the data, the source application, helps you understand the flow or direction in which data moves. For instance, RMS “owns” purchase orders, meaning you can only create purchase orders in RMS. For this reason, there is an RMS batch module that updates Oracle Financial Application 10.7 with purchase order data.

This table describes the data and ownership relationships. The graphic that follows provides further illustration:

Financial Data Ownership (Data flow from the “owner” application to the other application.)		
Kind of Data	Owned by	Updated to
Exchange Rates (Note: Currencies are manually entered into both Oracle and RMS. Only exchange rates are transferred via batch processing.)	Oracle AP	Retek Merchandising System (RMS)
Suppliers and Supplier Addresses (called “vendors” in Oracle)	Oracle AP and PO	RMS
Invoice pay site addresses	Oracle AP and PO	RMS
Freight and Payment Terms	Oracle AP and PO	RMS
Purchase Orders and Receipts (when not using Retek Invoice Matching)	RMS	Oracle AP and PO
Matched Invoices (when using Retek Invoice Matching)	Retek Invoice Matching (RIM)	Oracle AP
Stock Ledger (sales transaction data)	RMS	Oracle GL
Total values for a store’s daily sales, tenders, paid-ins, paid-outs, and taxes	Retek Sales Audit (ReSA)	Oracle GL



Currencies and Exchange Rates: from Oracle to RMS

During the implementation of the RMS and Oracle Financials interface, you manually enter the three-letter currency codes and exchange rate values into both applications. For later currency updates, you must enter these manually too. Currency rates, however, are updated by two batch modules.

First, the batch module FIFPDL107.PC runs to load Oracle currency rate data to the RMS staging table FIF_CURRENCY_RATES. FIFPDL107.PC accomplishes this using the argument in get_curr that first checks RMS system options for your financial application. Because the application is Oracle, in this case, the PL/SQL package ORA2RMS107b.pls actually loads Oracle currency rates to the staging table.

Finally, another batch module, FIFCURU2107.PC, runs to process the data from the staging table into RMS's CURRENCY_RATES table.

Suppliers and Supplier Addresses: from Oracle to RMS

RMS uploads suppliers from Oracle Financials for which there are RMS supplier types. You can view RMS' list of supplier types by typing this SQL*Plus statement:

Select * from code_detail where code_type = 'SUTY';

The update of suppliers and their addresses involves first loading data from Oracle to two RMS staging tables using one batch module and then processing that staged data to RMS tables using another batch module.

For the supplier update, the batch module FIFDLP107.PC runs to load Oracle supplier and supplier type data to the RMS staging table FIF_SUPS. Similar to the currency rate update (see earlier), FIFDLP107.PC uses the argument in get_vendors that first checks RMS system options to verify that Oracle Financial Application 10.7 is being used. Because the application is Oracle the PL/SQL package ORA2RMS107b.pls actually loads Oracle suppliers to the staging table. The batch module FIFVNDU2107.PC runs to validate suppliers and then to process the data from the staging table into RMS's SUPS table.

New for RMS 9.0: Support for Direct Store Delivery Expense Vendors as Partners

RMS 9.0 now supports direct store delivery—the delivery of merchandise to a store, by a supplier, or the delivery of services unique to the store, by an expense vendor, such as janitorial services—both of which are not the result of a centrally created purchase order. Deliverers of merchandise are called “suppliers” in RMS, and deliverers of services are called “partners”. Oracle considers both suppliers and expense vendors/partners to be “supplier types.” In addition to uploading merchandise vendors (suppliers) from Oracle to RMS's SUPS table, RMS 9.0 now uploads partner types for expense vendors to the PARTNER table. This now allows Retek Invoice Matching to match invoices for both direct store delivery suppliers and partners and to send both to Oracle AP for payment.

Supplier addresses are handled as part of the supplier update. The batch module FIFDLP107.PC runs to load Oracle supplier address data to the RMS staging table FIF_SUPS_ADD. FIFDLP107.PC uses the argument in get_vendors to first checks RMS system options for the financial application and loads the ORA2RMS107b.pls package that loads Oracle supplier addresses to the staging table. The batch module FIFVNDU2107.PC runs to process the data from the staging table into RMS's ADDR table.

New for RMS 9.0: Selecting the Invoice Pay Site

RMS 9.0 now supports the selection of the pay site address on invoices that Retek Invoice Matching sends to Oracle AP for payment. The pay site address is the address to where payment for an invoice is sent. Within Retek Invoice Matching, you can select the pay site on the invoice form. If you do not select the pay site, RIM will default to the first address found for that supplier or partner. You can set up the default address within Oracle before downloading all addresses to RMS (and RIM). The first address you set up becomes the default in the event that the user does not select another one.

Freight and Payment Terms: from Oracle to RMS

It is recommended that freight and payment terms not be edited in Oracle Financials. Instead it is preferable to create new terms. Freight terms have a restriction of two characters. Payment terms can have 15 characters. When uploading payment terms, RMS extracts only the first line from Oracle Financials' Payment Terms Lines. The update of terms begins with the batch module fifpdlp107.pc that runs with the argument in get_terms to load Oracle terms to the RMS staging table FIF_TERMS for payment terms and to FIF_FREIGHT_TERMS for freight terms.

The module FIFTRMU2107.PC then batch processes data from both staging tables to RMS's TERMS table for payment terms and FREIGHT_TERMS for freight terms. Any errors are written to IF_ERRORS.

Purchase Orders: from RMS to Oracle

In order to transfer purchase orders with order statuses of approved, canceled, or completed from RMS to Oracle, use the batch module ORDREV.PC.

Note: Use purchase order transfer batch module functionality described here only if you are not using Retek Invoice Matching. The invoice matching-Oracle process is described later in this document. However, ORDREV.PC should be executed daily regardless of the presence of the Oracle Financials interface.

ORDREV.PC loads RMS purchase orders from the ORDHEAD table to the staging table FIF_ORDHEAD. Then fifpdlp107.pc runs with the argument out get_order that calls the Oracle loading package RMS2ORA107b.pls to load the data to Oracle staging tables

Receipts: from RMS to Oracle

Note: RMS receipt data is accessed only as part of the purchase order loading process for determining which purchase orders are transferred.

The batch module PREPOST.PC, using the post_fifgldn1107 function loads the staging table FIF_RECEIVING with shipment receipts from RMS's IF_TRAN_DATA table.

Note: Use the PREPOST process for transferring receipts only if you are not using Retek Invoice Matching. The invoice matching-Oracle process is described in the next section.

Retek Invoice Matching (Invoice Matching) and Oracle Financials

If you are using Retek Invoice Matching, you can use it to transfer matched invoices to Oracle. Invoice Matching allows you to verify your received supplier shipments (receipts) against invoices and to confirm that the goods received are within cost and quantity tolerances that are defined for that supplier. You match invoices to supplier receipts (in the RMS SHIPMENT and SHIPSKU tables) by verifying goods received in a shipment and associated costs and then automatically approve invoices for payment. Once approved, you use two RMS batch modules to process them to Oracle staging tables.

RMS uses two batch modules to accomplish the loading of matched invoice data to Oracle staging tables:

- **INVCPOST.PC**—Processes new invoices that have a payment status of ‘H’ (hold payment) and ‘P’ (ready to pay), those that are approved for payment or are being force paid. This module also inserts prepaid supplier invoices of ‘U’ (prepaid but unmatched, if unapproved) and ‘M’ (prepaid and matched, if approved). Data is loaded to RMS staging tables IIF_HEAD, IIF_DETAIL, IIF_NON_MERCH, IIF_MERCH_VAT, and IIF_MATCH_DETAIL.
- **FIFINVCU107.PC**—Loads the data from the IIF_HEAD staging table to Oracle’s AP_EXPENSE_REPORT_HEADERS_ALL staging table. It also loads data from the other four IIF_ tables to Oracle’s AP_EXPENSE_REPORT_LINES_ALL table.

Stock Ledger (sales transaction data): from RMS to Oracle

Three RMS batch modules process sales transaction data from RMS’s stock ledger to RMS’s general ledger staging table FIF_GL_DATA, two for the daily transfer of transaction data to Oracle (from which you pick one), and one for the monthly transfer.

For daily sales transaction transfers to Oracle, you can choose one of two approaches. You can transfer sales transactions at the SKU/location level, or you can transfer sales transactions that are rolled up, or summarized, to the department/class/subclass level by location. The approach you use depends upon a system setting that you select in RMS’s system options during implementation of Oracle Financials. Thereafter, you run the daily batch module that matches that system setting. Here are descriptions for the two daily sales transaction batch modules and the batch process for each:

- **FIFGLDN1107.PC**—Processes daily sales by transaction types at the SKU and location level. Run this batch module if there are records in the CODE_DETAIL table with code_type value of GLST. When the batch module runs, it processes data from RMS’s IF_TRAN_DATA table.
- **FIFGLDN2107.PC**—Processes daily sales by transaction type that are rolled up to the department/class/subclass level by location. Run this batch module if the transaction code on the CODE_DETAIL table is. Run this batch module if there are records in the CODE_DETAIL table with code_type value of GLRT.

For processing monthly sales transaction data, use this batch module:

- FIFGLDN3107.PC—Processes monthly sales transactions that are rolled up to the department/class/subclass level by location. Run this batch module if the transaction code on the CODE_DETAIL table is GLMT. It send specific columns of the MONTH_DATA table (department/class/subclass level) to the Oracle GL. Run this batch module if there are records in the CODE_DETAIL table with a code_type value of GLMT.

After any of the general ledger batch modules loads the transaction data to the FIF_GL_DATA table, run FIFPDLP107.PC with the argument out get_gl_data to load the data to Oracle's GL_INTERFACE staging table.

How RMS Assigns Sales Transaction Data to Oracle

When you implement the Oracle Financials Application interface to RMS, you enter Oracle's chart of accounts into RMS's GL CROSS REFERENCE form (GLCROSSR107.FMB) form. This makes possible the assignment by RMS of transaction data in the stock ledger at the department/class/subclass level to specific account codes.

Retek Sales Audit Totals Export to Oracle General Ledger

In addition to RMS's stock ledger export to the Oracle general ledger, Retek Sales Audit (ReSA) also exports data to the general ledger. ReSA's SAEXPGL107.PC exports totals to the general ledger for:

- Sales transactions
- Tenders
- Paid-ins
- Paid-outs
- Taxes

This export batch module works by first getting the store day. Then it finds the totals to export and verifies that individual totals contain no errors. Next the module looks at the SA_FIF_ORA_CROSS_REF table to determine which Oracle account ID (CCID) it needs to export to. Finally it moves the data to the Oracle staging GL_INTERFACE. (No RMS staging table is involved.)

Be sure to read the note about the use of clearing accounts on the first page.

Summary of Oracle Financials Application Batch Modules

The following table lists the RMS batch modules used to process data to and from Oracle Financial Application 10.7.

Retek Batch Modules for Oracle Financials		
Batch Module Program Name	What It Does	Run Before/After Other Modules?
fifcuru2107.pc	Processes currency exchange rates from Oracle to RMS	Run after fifpdlp107.pc with the in get_curr argument
fifvndu2107.pc	Processes suppliers and supplier addresses from Oracle to RMS	Run after fifpdlp107.pc with the in get_vendors argument
fiftrmu2107.pc	Processes freight and payment terms from Oracle to RMS	Run after fifpdlp107.pc with the argument in get_terms
ordrev.pc	Processes purchase orders with approved, canceled, or completed order status to Oracle from RMS Ordrev.pc is always run for the processing of purchase order data, regardless of whether there is a financial interface present or not	Run before fifpdlp107.pc with the argument out get_order. (Used when Retek Invoice Matching is unavailable.)
prepost.pc (with the argument post-fifglnd107)	Processes shipment receipts to Oracle from RMS Used when Invoice Matching is unavailable	run after fifglnd1107.pc
invcpst.pc	Processes matched invoices to staging tables for output to Oracle Used when Invoice Matching is available	Run before fifinvcu107.pc
invmatch.pc	Matches invoices when Retek Invoicing Matching is available	Run before invcpst.pc
fifinvcu107.pc	Processes matched invoice data from RMS staging tables to Oracle staging tables	Run after invcpst.pc
fifglnd1107.pc	Processes daily sales transaction data at the SKU/location level from RMS to Oracle	Run before fifpdlp107.pc with the out get_gl_data argument
fifglnd2107.pc	Processes daily sales transaction data rolled up to the department/class/subclass level for location from RMS to Oracle	Run before fifpdlp107.pc with the out get_gl_data argument
fifglnd3107.pc	Processes monthly sales transaction data rolled up to department/class/subclass level for location from RMS to Oracle	Run before fifpdlp107.pc with the out get_gl_data argument

Chapter 13 - Overview of Retek Sales Audit 9.0 Batch Processing

Retek Sales Audit (ReSA) is another member of Retek Enterprise 9.0. ReSA offers a retailer the ability to quickly and accurately validate and audit sales transaction data before it is exported to other applications. ReSA uses several batch-processing modules to:

- Import sales transaction data to the ReSA database. The import process validates transactions and writes errors for reconciliation through the interactive (online) audit
- Produce totals from user-defined totaling calculation rules that a user can review during the interactive audit
- Validate transaction and total data with user-defined audit rules that generate errors whenever data has not met the criteria. The user can review these errors during the interactive audit
- Create and export files in formats suitable for transfer to other applications.
- Update the ReSA database with adjustments received from external systems on previously exported data

This document describes these processes and the batch processing modules involved with them.

Store Day Defined

The term *store day* is used throughout this document. Store day describes all sales transactions that occur in one business day at one store or location. Because customers need the ability to audit sales transactions on a store-by-store basis for a defined period of time, store day data are maintained separately beginning with the initial import of data from the point of sale system (POS). ReSA locks store day data until all import, validation, and totaling processes are complete.

Preparation for the Data Import

The first two batch modules run prior to the importing and processing of the sales transaction log(s) for a store day.

SASTDYCR.PC runs to prepare the database tables with information on data that are expected for import and export for each store on the following day. The module looks for all stores that are scheduled to be open the next day, those for which ReSA expects to receive a transaction log. Then the module creates a store day record in the ReSA store day sequence record for that business day and also creates records in the import and export log tables, as well as the flash sales tables. (A flash sales report shows sales for any time during the day.) SASTDYCR.PC also assures that no duplicate import and export records are created for a store day.

SAGETREF.PC retrieves the following data from the Retek Merchandising System (RMS) and ReSA databases and writes them to temporary reference files:

SKUs	UPCs	wastage	variable weight UPCs
store day	promotions	code types and codes	error codes and descriptions
credit card mappings	store POS mappings	tender type code and description	non-merchandise codes
partners	suppliers	cashiers	

The sales transaction import module (described in the next section) accesses these files when it validates store day data. Being able to read from reference files, instead of from the database itself, speeds the completion of the import and validation process.

Transaction Data Import and Validation

The next two batch modules:

- Import the sales transaction log from the POS
- Lock the store day records
- Validate transactions
- Check balances for over or under
- Verify the end of the store day's received sales transactions
- Unlock the store day records if all transactions for the day have been imported

Before the import module SAIMPTLOG.PC can begin to process a store day's sales transactions, it must receive a transaction log from the customer's POS that is in a Retek compatible file format, called RTLOG. The customer is responsible for converting its transaction logs to RTLOGs.

SAIMPTLOG.PC locks the store day records in the ReSA database and begins to validate the following sales transaction data:

store day combinations	transaction types	sub tran types	reason codes
SKUs	UPCs	promotions	credit card numbers
tender types	tender IDs	tax codes	item types
customer ID types	customer attribute types		

It also looks for duplicate or missing transaction numbers. Errors in transactions are written to error tables. Next the module looks for transactions that involve a voucher (gift certificates issued or redeemed or other credit vouchers). It writes those voucher transactions to a file for later processing by another batch module SAVOUCH.PC. Finally, SAIMPTLOG.PC produces output files that are loaded into ReSA's database, using the Oracle SQL*Load tool. SQL*Load is another timesaving technique that speeds the batch process.

ReSA Valid Transaction Types	
Transaction Code	Transaction Type
OPEN	Open
CLOSE	Close
COND	Daily Store Conditions
DCLOSE	Day close indicator
LOAN	Loan
METER	Meter Reading for Fuel
NOSALE	No Sale
PAIDIN	Paid In
PAIDOU	Paid Out
PULL	Pull
PUMPT	Pump Test for Fuel
PVOID	Post Void (A transaction that was rung later into the register to void something that occurred earlier at the same store/day. A post void updates the original transaction's sub-transaction type.)
REFUND	Return of customer's original check.
RETURN	Return
SALE	Sale
TANKDP	Tank Dip
TERM	Termination Record
TOTAL	POS generated totals
VOID	Void (aborted transaction)

It next marks the store day record in the ReSA import log as partially or fully loaded in the database by looking for a transaction type of DCLOSE. The import process is completed after SAIMPTLOGFIN.PC has updated the store, data and audit status of each store day record.

Total Calculations and Rules

ReSA automatically totals sales transactions based on calculation definitions that the customer's users have previously created using the online wizard. Whenever users create new calculation definitions or edit existing ones, they become part of the automated totaling process the next time that SATOTALS.PC runs.

Users also create auditing rules with the online wizard that ReSA applies to totals and other errors. Errors uncovered by these rules are available for review during the interactive audit. Like SATOTALS.PC, after users modify existing rules or create new ones, they become part of the rules the next time that SARULES.PC runs.

Export Store Day Transaction Data to Applications

ReSA can prepare data for export to applications after:

- All transactions for the day have been imported
- Totals have run
- Audit rules have run
- Errors in transactions and totals relevant for the system receiving the associated data are eliminated or overridden

ReSA uses separate batch modules to process export data to seven external applications. Depending upon the application, exported data consists of either transaction data or totals, or both. This table shows you the name of the application to which ReSA exports data, a description of the kind of data processed, and the ReSA batch module that processes data for that application:

Application Name	Data Exported	ReSA Batch Module Name
Retek Merchandising System (RMS)	Sales and return transactions	SAEXPRMS.PC
Retek Data Warehouse (RDW)	Transaction item details for: all sales returns paid-ins paid-outs voids post voids store conditions transactions Transaction tender details Store-level totals Cashier over or short totals.	SAEXPRDW.PC
Account Clearing House (ACH)	Bank deposit totals	SAEXPACH.PC
Reconciliation System (UAR-Driscoll)	Totals for: Lottery sales Bank deposits Money order totals Credit card totals	SAEXPUAR.PC
Oracle Site Fuels Management (SFM)	Metered sales totals Meter reading transactions Tank dip transactions	SAEXPSFM.PC
Oracle Financial Application General Ledger version 10.7	Totals for: Sales transactions Tenders Paid-ins Paid-outs Taxes	SAEXPGL107.PC
Retek Invoice Matching (RIM)	Invoice number Vendor number Payment reference number Proof of delivery number Payment date Paid indicator	SAEXPIM.PC

Transaction Data Exports and the Unit of Work

The process of exporting transaction data varies according to the unit of work selected in ReSA's system options. There are two units of work, transaction and store day. If the unit of work selection is transaction, ReSA exports transactions to RMS as soon as they are free of errors. If the unit of work selection is store day, transactions are not exported until all errors for that store day are either overridden or corrected.

Retek Merchandise System (RMS) Export

SAEXPRMS.PC transfers store day sales and return transaction data to RMS. This module rolls up transaction data to the SKU/store/day/pricepoint level. Next it writes the data to a file called POSU. This file is then available for upload by RMS's POSUPLD.PC batch module. If a ReSA user later modifies any transactions after the store day has been exported, SAEXPRMS.PC will note the flagged changes and re-export that data to RMS.

Retek Data Warehouse (RDW) Export

SAEXPRDW.PC writes four output files, one each for transaction item data, transaction tender data, store total data, and cashier or register total data. Each of these files is then made available to RDW batch module for upload into the data warehouse. If a ReSA user later modifies any transactions or totals after the store day has been exported, SAEXPRDW.PC notes the flagged changes and re-exports that data to RDW.

Oracle Financial Application 10.7 General Ledger Export

SAEXPGL107.PC exports totals to Oracle Financial Application 10.7's general ledger for:

- Sales transactions
- Tenders
- Paid-ins
- Paid-outs
- Taxes

This export batch module works by first getting the store day. Then it finds the totals to export and verifies that individual totals contain no errors. Next the module looks at the SA_FIF_ORA_CROSS_REF table to determine which Oracle account ID (CCID) it needs to export to. Finally it moves the data to the Oracle staging GL_INTERFACE. (No RMS staging table is involved.)

Account Clearing House (ACH) Export

SAEXPACH.PC produces anticipatory deposit totals for ACH processing. The next business day's deposit is estimated based upon the average of deposits for the same business day of the week for the past four weeks. The current day's actual deposit is compared to the estimated amount from the previous day, and the difference is added or subtracted from the estimated amount for the next day. SAEXPACH.PC formats deposit amounts to a standard BAI version 2 file for export to ACH. BAI is the Bank Administration Institute.

Universal Account Reconciliation System (UAR) Export

SAEXPUAR.PC selects lottery, bank deposit, money order, and credit card totals and writes them to output files for export to the J. Driscoll & Associates' UAR application. For each store day, the module posts all specified totals to their appropriate output files.

Note—Defining Totals for Oracle GL, ACH, and UAR

The customer needs to define totals to be exported to Oracle Financial Application's general ledger, ACH, and UAR using the online wizards. Totals described here are only examples of those that a customer might choose to define and later export.

Retek Invoice Matching (RIM) Export

SAEXPIM.PC provides invoicing support for Direct Store Delivery (DSD) by transferring transactions for invoices paid out at the store (that are imported by ReSA from the POS) to Retek Invoice Matching. RIM then uses that data to create an invoice for DSD within RIM. Data exported to RIM by this batch module are:

- Invoice number
- Vendor number
- Payment reference number
- Proof of delivery number
- Payment date
- Paid indicator

Note: Retek Invoice Matching matches unpaid invoices that are marked as approved and sends them to Oracle Financial Application 10.7's Accounts Payable for payment.

Escheatment Totals to RIM for Accounts Payable

The laws of individual states and countries require a retailer to return monies for aged, unclaimed gift certificates and vouchers. This is called “escheatment.” SAESCHEAT.PC writes records for this data to tables that are read into RIM by SAEXPIM.PC. The data can then be sent as invoices approved for payment to the Oracle Financial Application Accounts Payable.

Oracle Site Fuels Management (SFM) Export

SAEXPSFM.PC exports the following data to Oracle's Site Fuels Management (SFM) application:

- Pump test transactions
- Metered sales totals
- Meter reading transactions
- Tank dip transactions

The SFM export process differs from the other four exports described earlier. Any store day that the module cannot lock is reported to an error log, and the module proceeds to the next store day. In addition, the module passes the currency code for the exported totals or transactions to SFM.

Another batch module, SAMASTERSFM.PC, transfers ReSA and RMS reference data to SFM for use during the fuels audit.

Full Disclosure and Post-Export Changes

If a user modifies data during the interactive audit that was previously exported to RMS or RDW, ReSA export batch modules re-export the modified data in accordance with a process called *full disclosure*. Full disclosure means that any previously exported values (dollars, units, and so on) are fully backed out before the new value is sent. Here is an example. Suppose that a transaction originally shows a sale of 12 items and that this transaction is exported. Later, during the interactive audit, a user determines that the correct amount is 15 items (three more than the original amount) and makes the change. ReSA then flags the corrected amount for export to the application.

Now during the export process, instead of simply adding three items to that transaction (which would change the amount from 12 to 15), a minus 12 (-12) is sent to back out the original amount of 12. Then an amount of 15 is sent. The result is that a transaction is corrected by fully accounting for the original amount before adding the correct one. Full disclosure, then, is meant to completely account for all adjustments.

What Happens to Totals when Transactions Are Modified?

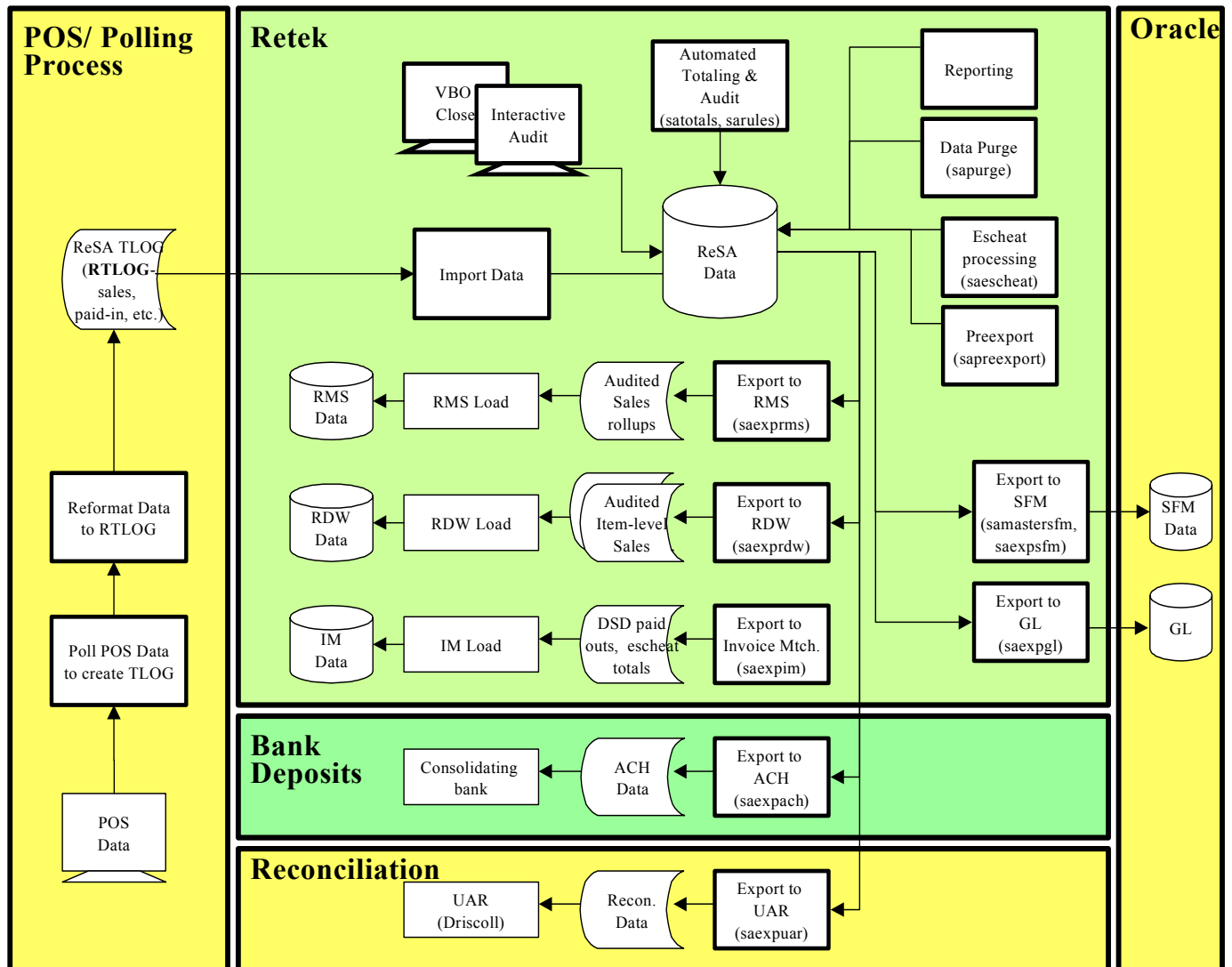
If a user modifies transactions during the ReSA interactive audit process, the totaling and auditing processes run again to recalculate store day totals. The batch module SAPREEXPORT.PC tracks all changed totals for the store day since the last export. The module writes the changes to revision tables that the export modules later recognize as ready for export.

Adjustments Received from an Application

Whenever an application, such as SFM or UAR, returns an adjustment to a total previously received from ReSA, a package is called from either the SFM form itself or the batch module SAIMPADJ.PC. This package is responsible for updating ReSA with the change. Upon receiving the adjustment from the application, the module identifies the total in the store day record that was exported. A revision of this total is created with the revised data. Totaling and auditing run to recalculate store day totals. New records are created for export batch modules that send adjusted data to applications, except for the one that provided the adjustment. SAIMPADJ.PC runs after the ReSA transaction import process and before the totaling process.

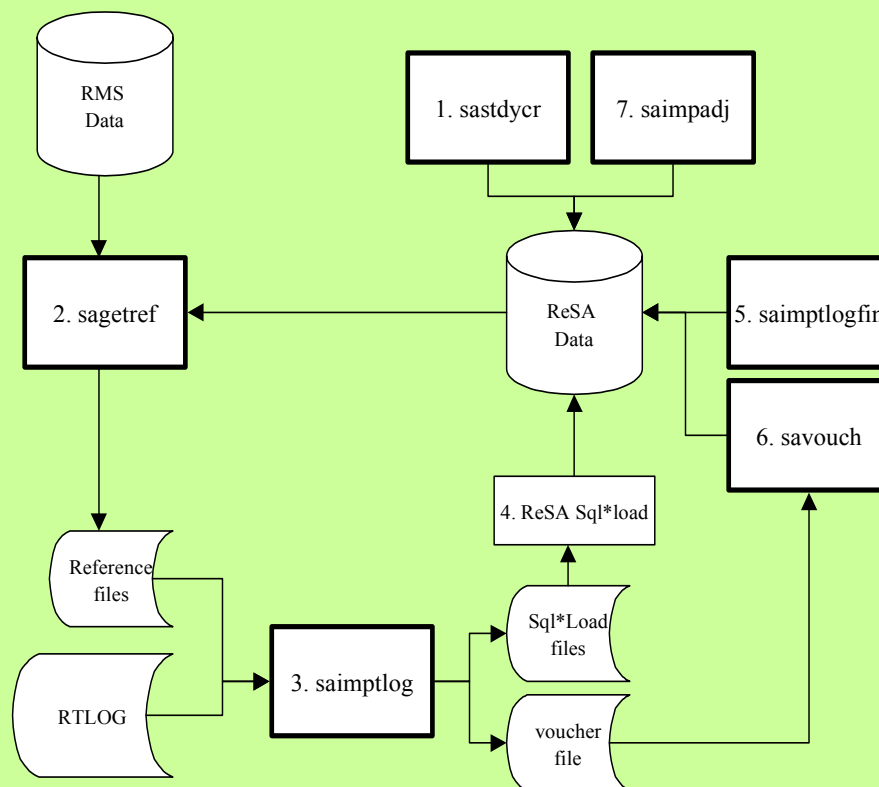
Chapter 14 - Retek Sales Audit Dataflow Diagrams

The following two diagrams illustrate how data flows within ReSA and between ReSA and other applications. Retek Sales Audit Dataflow Process shows the entire process, and Retek Sales Audit Import Process Dataflow expands the description of the Import Data block of the first diagram.



Retek Sales Audit Dataflow Process

Import Data



Retek Sales Audit Import Process Dataflow

Summary of ReSA Batch Modules

The following table lists the ReSA batch modules that are involved with processing POS sales transaction data, audit totals and rules, exports to other applications, and modifications and adjustments.

Retek ReSA Batch Modules		
Batch Module Program Name	What It Does	Run Before/After Other Modules?
sastdycr.pc	Sets up store day tables in anticipation of data import from the POS	Run before the next store day's transactions are received
sagetref.pc	Fetches reference data used during the import and validation process	Run daily before saimptlog.pc
saimptlog.pc	Imports and processes RTLOGs. Validates transactions and writes errors for reconciliation through the interactive (online) audit 9.0 supports DSD transactions for vend #, vend inv #, pay ref #, proof del. #	Run after sagetref.pc Run before saimptlogfin.pc Run multiple times daily in trickle polling environments
saimptlogfin.pc	Creates balances (over or under) by store, register, or cashier. Marks the store day record in the ReSA import log as partially or fully loaded. Unlocks store day records after all store transactions are imported	Run after saimptlog.pc Run before satotals.pc Run multiple times daily in trickle polling environments
satotals.pc	Produces totals from user-defined total calculation rules	Run after saimptlogfin.pc Run before sarules.pc
sarules.pc	Processes system and functional errors for viewing by users during the interactive audit	Run after satotals.pc Run before saescheat.pc
saescheat.pc	Sends escheatment totals for each state or country as defined by the retailer to Retek Invoice Matching	Run after sarules.pc Run before saexpim.pc
sapreexport.pc	Tracks changes in totals previously exported. Writes changes to revision tables recognized by the export modules for re-export to applications	Run before all export modules that require totals

Retek ReSA Batch Modules		
Batch Module Program Name	What It Does	Run Before/After Other Modules?
saexpsfm.pc	Transfers store day fuel and fuel tank transactions to the Oracle Site Fuels Management application	Run after sapreexport.pc
samastersfm.pc	Sends reference (stores, items) data from ReSA to Oracle Site Fuels Management	Run before saexpsfm.pc
saexprms.pc	Transfers imported, validated store day sales transactions to a POSU file for export to RMS	Run after sapreexport.pc
saexprdw.pc	Exports transactions and total data to RDW	Run after sapreexport.pc
saexpach.pc	Produces estimated store day deposit totals and formats totals in a standard BAI format file for export to ACH	Run after sapreexport.pc
saexpuar.pc	Writes totals for lottery, bank deposit, money order, and credit card to output files for export to the Driscoll UAR application	Run after sapreexport.pc
saimpadj.pc	Processes adjustments from applications. Creates new records for all export modules that transfer adjusted data, with the exception of the application that provided the adjustment	Run after saimptlog.pc. Run before satotals.pc
savouch.pc	Processes voucher transactions (e.g., gift certificates)	Run after saimptlog.pc
saexpgl107.pc	Exports totals to Oracle Financial Application version 10.7 General Ledger for: sales transactions tenders paid-ins paid-outs taxes	Run after satotals.pc and sarules.pc
saexpim.pc	Exports to Retek Invoice Matching: Invoice number Vendor number Payment reference number Proof of delivery number Payment date Paid indicator	Run after saescheat.pc Run before sapurge.pc

Retek ReSA Batch Modules		
Batch Module Program Name	What It Does	Run Before/After Other Modules?
sapurge.pc	Purges data that: Is older than the number of days that you indicate in ReSA's system options Has no errors Is not currently locked by another system	

ReSA 9.0 RTLOG Layout

The following illustrates the file layout format of the Retek TLOG. The content of each Retek TLOG file is per store per day. The filename convention will be RTLOG_STORE_DATETIME.DAT (e.g. RTLOG_1234_01221989010000.DAT)

FHEAD	Only 1 per file, required.
THEAD	Multiple expected, one per transaction, required for each transaction.
TCUST	Only 1 per THEAD record allowed, optional for some transaction types, see table below.
CATT	Attribute record specific to the TCUST record – Multiple allowed, only valid if TCUST exists.
TITEM	Multiple allowed per transaction, optional for some transaction types, see table below.
IDISC	Discount record specific to the TITEM record – Multiple allowed per item, optional see table below.
TTAX	Multiple allowed per transaction, optional see table below.
TTEND	Multiple allowed per transaction, optional for some transaction types, see table below.
TTAIL	1 per THEAD, required.
FTAIL	1 per file, required.

The order of the records within the transaction layout above is important. It aids processing by ensuring that information is present when it is needed.

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	File Type Definition	Char(4)	RTLGL	Identifies file as 'Retek TLOG'.	Y	Left/Blank
	File Create Date	Char(14)	Create date	Date and time file was written by external system (YYYYMMDDHHMMSS).	Y	Left/None

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	Business Date	Char(8)	Business Date to process	Business date of transactions. (YYYYMMDD).	Y	Left/None
	Location Number	Char(4)	Specified by external system	Store or warehouse identifier.	Y	Left/None
	Reference Number	Char(30)	Specified by external system	This may contain the Polling ID associated with the consolidated TLOG file or used for other purpose.	N	Left/Blank
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type.	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Register	Char(5)		Till used at store.	Y	Left/Blank
	Transaction Date	Char(14)	Transaction date	Date transactions were processed at the POS (YYYYMMDDHHMMSS).	Y	Left/None
	Transaction Number	Number(10)		Transaction identifier.	Y	Right/0
	Cashier	Char(10)		Cashier identifier.	N	Left/Blank
	Salesperson	Char(10)		Salesperson identifier.	N	Left/Blank
	Transaction Type	Char(6)	Refer to 'TRAT' code_type for a list of valid types.	Transaction type.	Y	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	Sub-transaction type	Char(6)	Refer to 'TRAS' code_type for a list of valid types.	Sub-transaction type. For sale, it can be employee, drive-off etc.	N	Left/Blank
	Orig_tran_no	Number(10)		Populated only for post-void transactions. Transaction number for the original tran that will be cancelled.	N	Right/0
	Orig_reg_no	Char(5)		Populated only for post-void transactions. Register number from the original tran.	N	Left/Blank
	Reason Code	Char(6)	Refer to 'REAC' code_type for a list of valid codes. If the transaction type is 'PAIDOU' and the sub transaction type is 'MV' or 'EV' then the valid codes come from the non_merch_code_head table.	Reason entered by cashier for some transaction types. Required for Paid In and Paid out transaction types, but can also be used for voids, returns, etc.	N	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	Vendor Number	Char(10)		Supplier id for a merchandise vendor paid out transaction, partner id for an expense vendor paid out transaction.	N	Left/Blank
	Vendor Invoice Number	Char(30)		Invoice number for a vendor paid out transaction.	N	Left/Blank
	Payment Reference Number	Char(16)		The reference number of the tender used for a vendor payout. This could be the money order number, check number, etc.	N	Left/Blank
	Proof of Delivery Number	Char(30)		Proof of receipt number given by the vendor at the time of delivery. This field is populated for a vendor paid out transaction.	N	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	Reference Number 1	Char(30)		Number associated with a particular transaction, for example weather for a Store Conditions transaction.	N	Left/Blank
	Reference Number 2	Char(30)		Second generic reference number.	N	Left/Blank
	Reference Number 3	Char(30)		Third generic reference number.	N	Left/Blank
	Reference Number 4	Char(30)		Fourth generic reference number.	N	Left/Blank
	Value Sign	Char(1)	Refer to 'SIGN' code_type for a list of valid codes.	Sign of the value.	Y if Value is present	Left/None

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	Value	Number(20)	Value with 4 implied decimal places. Populated by the retailer for TOTAL trans, populated by Retek sales audit for SALE, RETURN trans.		Y if tran is a TOTAL.	Right/0
Transaction Customer	File Type Record Descriptor	Char(5)	TCUST	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Customer ID	Char(16)	Customer identifier	The ID number of a customer.	Y	Left/Blank
	Customer ID type	Char(6)	Refer to 'CIDT' code_type for a list of valid types	Customer ID type.	Y	Left/Blank
	Customer Name	Char(40)		Customer name.	N	Left/Blank
	Address 1	Char(40)		Customer address.	N	Left/Blank
	Address 2	Char(40)		Additional field for customer address.	N	Left/Blank
	City	Char(30)		City	N	Left/Blank
	State	Char(3)	State identifier	State.	N	Left/Blank
	Zip Code	Char(10)	Zip identifier	Zip code.	N	Left/Blank
	Country	Char(3)		Country.	N	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	Home Phone	Char(20)		Telephone number at home.	N	Left/Blank
	Work Phone	Char(20)		Telephone number at work.	N	Left/Blank
	E-mail	Char(100)		E-mail address.	N	Left/Blank
	Birthdate	Char(8)		Date of birth. (YYYYMMDD)	N	Left/Blank
Customer Attribute	File Type Record Descriptor	Char(5)	CATT	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Attribute type	Char(6)	Refer to 'SACA' code_type for a list of valid types	Type of customer attribute	Y	Left/Blank
	Attribute value	Char(6)	Refer to members of 'SACA' code_type for a list of valid values	Value of customer attribute.	Y	Left/Blank
Transaction Item	File Type Record Descriptor	Char(5)	TITEM	Identifies file record type.	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	Item Status	Char(6)	Refer to 'SASI' code_type for a list of valid codes.	Status of the item within the transaction, V for item void, S for sold item, R for returned item.	Y	Left/Blank
	Item Type	Char(6)	Refer to 'SAIT' code_type for a list of valid codes.	Identifies what type of item is transmitted.	Y	Left/Blank
	SKU	Number(8)	Item identifier	ID number	Either SKU	Left/Blank
	UPC	Char(13)	Item identifier	ID number	Or UPC	Left/Blank
	Supplement	Number(5)	Supplemental identifier	Used to further specify the ID of a UPC.	N	Left/Blank
	Voucher	Char(16)		Gift certificate number	N	Right/0
	Item Number	Char(16)	Item identifier	Populated by retailer for Item types other than SKU, UPC or GCN. Allows retailers more flexibility to store additional item types within ReSA.	N	Left/Blank
	Department	Number(4)		Identifies the department this item belongs to. This is filled in by saimptlog.	N	Right/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	Class	Number(4)	Item's class	Class of item sold or returned. Not required from a retailer, populated by Retek sales audit.	N	Right/Blank
	Subclass	Number(4)	Item's subclass	Subclass of item sold or returned. Not required from a retailer, populated by Retek sales audit. This is filled in by saimptlog.	N	Right/Blank
	System Indicator	Char(1)	Refer to 'IMTP' code_type for a list of valid codes.	The type of item sold or returned. Not required from a retailer, populated by Retek sales audit. This is filled in by saimptlog.	N	Left/None
	Quantity Sign	Char(1)	Refer to 'SIGN' code_type for a list of valid codes.	Sign of the quantity	Y	Left/None
	Quantity	Number(12)		Number of items purchased with 4 decimal places.	Y	Right/0

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	Unit Retail	Number(20)		Unit retail with 4 implied decimal places	Y	Right/0
	Override Reason	Char(6)	Refer to 'ORRC' code_type for a list of valid codes.	This column will be populated when an item's price has been overridden at the POS to define why it was overridden.	Y if unit retail was manually entered	Left/Blank
	Original Unit Retail	Number(20)		Value with 4 implied decimal places. This column will be populated when the item's price was overridden at the POS and the item's original unit retail is known.	Y if unit retail was manually entered	Right/0
	Taxable Indicator	Char(1)	Refer to 'YSNO' code_type for a list of valid codes.	Indicates whether or not item is taxable.	Y	Left/None
	Pump	Char(8)		Fuel pump identifier.	N	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	Reference Number 5	Char(30)		Number associated with a particular item within a transaction, for example special order number.	N	Left/Blank
	Reference Number 6	Char(30)		Second generic reference number at the item level.	N	Left/Blank
	Reference Number 7	Char(30)		Third generic reference number at the item level.	N	Left/Blank
	Reference Number 8	Char(30)		Fourth generic reference number at the item level.	N	Left/Blank
	Item_swiped_ind	Char(1)	Refer to 'YSNO' code_type for a list of valid codes.	Indicates if the item was automatically entered into the POS system or if it had to be manually keyed.	Y	Left/None
	Return Reason Code	Char(6)	Refer to 'SARR' code_type for a list of valid codes.	The reason an item was returned.	N	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	Salesperson	Char(10)		The salesperson who sold the item.	N	Left/Blank
	Expiration_date	Char(8)		Gift certificate expiration date (YYYYMMDD).	N	
Item Discount	File Type Record Descriptor	Char(5)	IDISC	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	RMS Promotion Number	Char(6)	Refer to 'PRMT' code_type for a list of valid types	The RMS promotion type.	Y	Left/Blank
	Discount Reference Number	Number(4)		Discount reference number is associated with the discount type (e.g. if discount type is a promotion, this contains the promotion number).	N	Left/Blank
	Discount Type	Char(6)	Refer to 'SADT' code_type for a list of valid types.	The type of discount within a promotion. This allows a retailer to further break down coupon discounts within the "In-store" promotion, for example.	N	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	Coupon Number	Char(16)		Number of a store coupon used as a discount.	Y if coupon	Left/Blank
	Coupon Reference Number	Char(16)		Additional information about the coupon, usually contained in a second bar code on the coupon.	Y if coupon	Left/Blank
	Quantity Sign	Char(1)	Refer to 'SIGN' code_type for a list of valid codes.	Sign of the quantity.	Y	Left/None
	Quantity	Number(12)		The quantity purchased that discount is applied with 4 implied decimal places.	Y	Right/0
	Unit Discount Amount	Number(20)		Unit discount amount for this item with 4 implied decimal places.	Y	Right/0
	Reference Number 13	Char(30)		Number associated with a particular transaction type at the discount level. The sa_reference table defines what this field can contain for each transaction type.	N	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	Reference Number 14	Char(30)		Second generic reference number at the discount level.	N	Left/Blank
	Reference Number 15	Char(30)		Third generic reference number at the discount level.	N	Left/Blank
	Reference Number 16	Char(30)		Fourth generic reference number at the discount level.	N	Left/Blank
Transaction Tax	File Type Record Descriptor	Char(5)	TTAX	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Tax Code	Char(6)	Refer to 'TAXC' code_type for a list of valid codes	Tax code to represent whether it is a state tax type, provincial tax, etc.	Y	Left/Blank
	Tax Sign	Char(1)	Refer to 'SIGN' code_type for a list of valid codes.	Sign of Tax Amount.	Y	Left/None
	Tax Amount	Number(20)		Amount of tax charged for this tax code type in a transaction with 4 implied decimal places.	Y	Right/0
Transaction Tender	File Type Record Descriptor	Char(5)	TTEND	Identifies file record type	Y	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Tender Type Group	Char(6)	Refer to 'TENT' code_type for as list of valid types	High-level grouping of tender types.	Y	Left/Blank
	Tender Type ID	Number(6)	Refer to the pos_tender_type_head table for as list of valid types	Low-level grouping of tender types.	Y	Left/Blank
	Tender Sign	Char(1)	Refer to 'SIGN' code_type for a list of valid codes.	Sign of the value.	Y	Left/None
	Tender Amount	Number(20)		Amount paid with this tender in the transaction with 4 implied decimal places.	Y	Right/0
	Cc_no	Number(16)		Credit card number	Y if credit card	Left/Blank
	Cc_auth_no	Char(16)		Authorization number for a cc	Y if credit card	Left/Blank
	cc authorization source	Char(6)	Refer to 'CCAS' code_type for as list of valid types		Y if credit card	Left/Blank
	cc cardholder verification	Char(6)	Refer to 'CCVF' code_type for as list of valid types		Y if credit card	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	cc expiration date	Char(8)		(YYYYMMDD)	Y if credit card	Left/Blank
	cc entry mode	Char(6)	Refer to 'CCEM' code_type for as list of valid types	Indicates whether the credit card was swiped, thus automatically entered, or manually keyed.	Y if credit card	Left/Blank
	cc terminal id	Char(6)		Terminal number transaction was sent from.	N	Left/Blank
	cc special condition	Char(6)	Refer to 'CCSC' code_type for as list of valid types		Y if credit card	Left/Blank
	Voucher_no	Char(16)		Gift certificate or credit voucher serial number.	Y if voucher	Right/0
	Coupon Number	Char(16)		Number of a manufacturer's coupon used as a tender.	Y if coupon	Left/Blank
	Coupon Reference Number	Char(16)		Additional information about the coupon, usually contained in a second bar code on the coupon.	Y if coupon	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	Reference No 9	Char(30)		Number associated with a particular transaction type at the tender level.	N	Left/Blank
	Reference No 10	Char(30)		Second generic reference no at the tender level.	N	Left/Blank
	Reference No 11	Char(30)		Third generic reference no at the tender level.	N	Left/Blank
	Reference No 12	Char(30)		Fourth generic reference no at the tender level.	N	Left/Blank
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type	Y	Left/Blank
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	Transaction Record Counter	Number(10)		No of records processed in current tran (only records between trans head & tail)		
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type	Y	Left/Blank

Record Name	Field Name	Field Type	Default Value	Description	Required?	Justification/ Padding
	File Line Identifier	Number(10)	Specified by external system	ID of current line being processed by input file.	Y	Right/0
	File Record Counter	Number(10)		No of transactions processed in current file (only records between file head & tail)	Y	Right/0

The RTLOG file is imported into the Sales Audit tables after validation by the batch program `saimptlog`. This section describes the requirements and validations performed on the records.

1. Common requirements/validations:

This section details the common requirements and validations performed on all transactions. The following sections describe the specific requirements of each type of transaction. If a transaction is not mentioned, then it does not have specific requirements.

a. Record Type Requirements:

Transaction Type	Includes item records?	Includes tender records?	Includes tax records?	Includes customer records?
OPEN	No	No	No	No
NOSALE	No	Optional	No	No
VOID	Optional	Optional	Optional	Optional
PVOID	No	No	No	No
SALE	Yes	Yes	Optional	Optional
RETURN	Yes	Yes	Optional	Optional
EEXCH	Yes	No	Optional	Optional
PAIDIN	No	Yes	No	No
PAIDOU	No	Yes	No	No
PULL	No	Yes	No	No
LOAN	No	Yes	No	No
COND	No	No	No	No
CLOSE	No	No	No	No
TOTAL	No	No	No	No

Transaction Type	Includes item records?	Includes tender records?	Includes tax records?	Includes customer records?
REFUND	This transaction is not sent through the RTLOG. It is entered at the HQ level. The TITEM and TCUST records are optional. The TTEND record is required. A TTAX record should not be included.			
METER	Yes	No	No	No
PUMPT	Yes	No	No	No
TANKDP	Yes	No	No	No
TERM	TERM records are created by saimptlog and then loaded into the database. They do not come from the RTLOG file. They require one TITEM, one TTEND, one TTAX, one TCUST record and one CATT record.			
DCLOSE	No	No	No	No

b. Requirements per record type:

Record Type	Requirements
IDISC	IDISC records must immediately follow their associated TITEM record.
CATT	CATT records must immediately follow their associated TCUST record.

c. Code Type Validations:

Record Name	Field Name	Code Type
Transaction Header	Transaction Type	TRAT
	Sub-transaction Type	TRAS
	Reason Code	REAC or values from non_merch_code_head if the transaction type is 'PAIDOU' and the sub transaction type is 'MV' or 'EV'.
	Value Sign	SIGN
	Vender No	If the transaction type is 'PAIDOU' and the sub transaction type is 'MV', this field is validated against the supplier table. If the transaction type is 'PAIDOU' and the sub transaction type is 'EV', this field is validated against the partner table.

Record Name	Field Name	Code Type
Transaction Item	Item Type	SAIT
	Item Status	SASI
	System Indicator	IMTP
	Quantity Sign	SIGN
	Taxable Indicator	YSNO
	Price Override Reason Code	ORRC
	Item Swiped Indicator	YSNO
	Return Reason Code	SARR
Item Discount	RMS Promotion Type	PRMT
	Discount Type	SADT
	Quantity Sign	SIGN
Transaction Customer	Customer ID Type	CIDT
Customer Attribute	Attribute Type	SACA
	Attribute value	Code types from codes in SACA.
Transaction Tax	Tax code	TAXC
	Tax sign	SIGN
Transaction Tender	Tender Type Group	TENT
	Tender Sign	SIGN
	Tender Type ID	Pos_tender_type_head table
	CC Authorization Source	CCAS
	CC Cardholder Verification	CCVF
	CC Entry Mode	CCEM
	CC Special Condition	CCSC

- d. Dates are validated: Business Date, Transaction Date, Expiration Date Also, saimptlog accepts only business dates that are within the PERIOD.VDATE minus the SA_SYSTEM_OPTIONS.DAYS_POST_SALE value.
- e. Store number is validated against the STORE table.
- f. Numeric fields are checked for non-numeric characters.

- g. For transaction of type SALE, RETURN and EEXCH, saimptlog checks whether a transaction is in balance:

Transaction Items (Unit Retail * Unit Retail Sign * Quantity)

+ Item Discounts (Unit Discount Amount * Unit Discount Sign * Quantity)

+ Transaction Tax (Tax Amount * Tax Sign)

= Transaction Tenders (Tender Amount * Tender Sign)

saimptlog will populate the Value field (on THEAD) with the transaction's sales value (item value – discount value + tax value) from the above calculation if it was not provided in the RTLOG.

- h. Treatment of vouchers.

If an item sold is a gift certificate (Transaction Item, Voucher field has a value), issued information is written to the SA_VOUCHER table.

If the Transaction Type is a RETURN, and the Transaction Tender Type Group is voucher (VOUCH), issued information is written to the SA_VOUCHER table.

If the Transaction Type is a SALE, and the Transaction Tender Type Group is a voucher (VOUCH), redeemed information is written to the SA_VOUCHER table.

When a gift certificate is sold, customer information should always be included. A receiving customer name value should be populated in the ref_no5 field, a receiving customer state value should be populated in the ref_no6 field and a receiving customer country should be populated in the ref_no7 field. These reference fields can be changed by updating the sa_reference table *but the code needs to be modified too*. The expiration date is put on the expiration_date field on the TITEM record.

- i. Other validations/points of interest:

A salesperson in the TITEM record takes precedence over the salesperson in the THEAD record.

If an item sold is a UPC (Transaction Item, UPC field has a value and SKU does not), it will be converted to the corresponding SKU using the Supplement.

If an item sold is a SKU (Transaction Item, SKU field has a value), it will be validated against RMS item tables.

The corresponding Department, Class, Subclass, System Indicator and Taxable Indicator will be selected from the RMS tables and populated for a SKU.

- j. The balancing level determines whether the register or the cashier fields are required.

If the balancing level is 'R'egister, then the register field on the THEAD must be populated.

If the balancing level is 'C'ashier, then the cashier field on the THEAD must be populated.

If the balancing level is 'S'tore, then neither field is required to be populated.

- k. The tax_ind and the item_swiped_ind fields can only accept 'Y' or 'N' values. If an invalid value is passed through the RTLOG, an error will be flagged and the value will be defaulted to 'Y'.

Transaction of type 'SALE':

A transaction of type SALE is generated whenever an item is sold. A sale may be to an employee, the sub-transaction type would be EMP in this case. Or it may be a drive-off sale (sub-transaction type DRIVEO) when someone drives off with unpaid gas. A special type of sale is an "odd exchange" (sub-transaction type EXCH) where items are sold and returned in the same transaction. If the net value of the exchange is positive, then it is a sale. If the net value is negative, it is a return. If the net value is zero and the items exchanged are in the same SKU style, it would be a transaction of type EEXCH (Even Exchange).

2. Requirements per record type (other than what is described in Layout section above):

Record Type	Requirements
THEAD	
TITEM	Item Status is a required field; it determines whether the item is 'S'old, 'R'eturned or 'V'oided. If the item status is S, the quantity sign is expected to be P. If the item status is 'R', the quantity sign is expected to be N. If the item status is V, the quantity sign is the reverse of the quantity sign of the voided item. That is, if an item with status S is voided, the quantity sign would be N. Furthermore, the sum of the quantities being voided cannot exceed the sum of the quantities 'S'old or 'R'eturned. Note: neither of the above two validations are performed by saimptlog but an audit rule could be created to check this. In a typical sale, the items would all have a status of 'S'. In the case of an odd exchange, some items will have a status of 'R'. In a typical return, the items would all have a status of 'R'. In the case of an odd exchange, some items will have a status of 'S'. If an item has status R, then the Return Reason Code field may be populated. If it is, it will be validated against code type 'SARR'. If the price of an item is overridden, then the Override Reason and Original Unit Retail fields must be populated.

Record Type	Requirements
IDISC	The RMS Promotion Type field must always be populated with values of code type 'PRMT'.
TTEND	If the tender type group is 'COUPON', then the Coupon Number field must be populated. The Coupon Reference Number field is optional. If the Transaction Tender Type Group is a credit card (CCARD), the number will be validated against the SA_CC_VAL table. The other cc fields are optional.

a. Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Transaction Type	Sub-transaction Type	Item Type	Tender Type Group	Reference Number Field	Meaning of Reference Field	Req?
SALE				1	Speed Sale Number	Y
SALE		GCN		5	Recipient Name	N
SALE		GCN		6	Recipient State	N
SALE		GCN		7	Recipient Country	N
SALE			CHECK	9	Check Number	N
SALE			CHECK	10	Driver's License Number	N
SALE			CHECK	11	Credit Card Number	N
SALE	DRIVEO			1	Incident Number	Y
SALE	EMP			3	Employee Number of the employee receiving the goods.	N

b. Expected values for sign fields

TRANSACTION TYPE	TITEM.Quantity Sign	TTEND.Tender Sign	TTAX.Tax Sign	IDISC.Quantity Sign
SALE	P if item is sold; N if item is returned; reverse of original item if item is voided.	P	P	P if item is sold; N if item is returned; reverse of original item if item is voided.

3. Transaction of type 'PVOID':

This transaction is generated at the register when another transaction is being post voided. The orig_tran_no and orig_reg_no fields must be populated with the appropriate information for the transaction being post voided. The PVOID transaction must be associated with the same store day as the original transaction. If the PVOID needs to be generated after the store day is closed, the transaction needs to be created using the forms.

4. Transaction of type 'RETURN':

This transaction is generated when a customer returns an item.

- a. This type of transaction has similar record type requirements as a 'SALE' transaction.
- b. Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Transaction Type	Sub-transaction Type	Reference Number Field	Meaning of Reference Field	Req?
RETURN		1	Receipt Indicator (Y/N)	Y
RETURN		2	Refund Reference Number	N
RETURN	EMP	3	Employee Number of the employee returning the goods.	N

c. Expected values for sign fields

TRANSACTION TYPE	TITEM.Quantity Sign	TTEND.Tender Sign	TTAX.Tax Sign	IDISC.Quantity Sign
RETURN	P if item is sold; N if item is returned; reverse of original item if item is voided.	N	N	P if item is sold; N if item is returned; reverse of original item if item is voided.

5. Transaction of type 'EEXCH':

This transaction is generated when there is an even exchange.

- This type of transaction has similar record type requirements as a 'SALE' transaction.
- It is expected that the number of items returned equals the number of items sold. However, this validation is not performed by saimptlog. An audit rule could be created for this. Saimptlog only expects that there would be at least two item records.
- No tender changes hands in this transaction.
- Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Transaction Type	Sub-transaction Type	Reference Number Field	Meaning of Reference Field	Req?
EEXCH		1	Receipt Indicator (Y/N)	Y
EEXCH	EMP	3	Employee Number of the employee exchanging the goods.	N

6. Transaction of type 'PAIDIN':
- This type of transaction has only one TTEND record.
 - A reason code is required.
 - Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Reason Code	Reference Number Column	Meaning	Req?
NSF	1	NFS Check Credit Number	N
ACCT	1	Account Number	N

7. Transaction of type 'PAIDOU':
- This type of transaction has only one TTEND record.
 - A reason code is required (code type REAC). If the sub-transaction type is 'EV' or 'MV', the reason code comes from the non_merch_codes_head table.
 - If the sub-transaction type is 'EV' or 'MV', then at least one field among the vendor number, vendor invoice number, payment reference number and proof of delivery number fields should be populated.
 - If the sub-transaction type is 'EV', then the vendor number comes from the partner table. If the sub-transaction type is 'MV', then the vendor number comes from the supplier table.
 - Meaning of reference number fields:

Notes: The meaning of these reference number fields may be changed through the sa_reference table.

Sub Transaction Type	Reason Code	Reference Number Column	Meaning	Req?
EV		2	Personal ID Number	N
EV		3	Routing Number	N
EV		4	Account Number	N
	PAYRL	1	Money Order Number	N
	PAYRL	2	Employee Number	N
	INC	1	Incident Number	N

8. Transaction of type 'PULL':

This transaction is generated when cash is withdrawn from the register.

a. This type of transaction has only one TTEND record.

b. Expected values for sign fields

TRANSACTION TYPE	TITEM.Quantity Sign	TTEND.Tender Sign	TTAX.Tax Sign	IDISC.Quantity Sign
PULL	N/A	N	N/A	N/A

9. Transaction of type 'LOAN':

This transaction is generated when cash is added to the register.

a. This type of transaction has only one TTEND record.

b. Expected values for sign fields

TRANSACTION TYPE	TITEM.Quantity Sign	TTEND.Tender Sign	TTAX.Tax Sign	IDISC.Quantity Sign
LOAN	N/A	P	N/A	N/A

10. Transaction of type 'COND':

This transaction records the condition at the store when it opens. There can be at most one COND record containing weather information and at most one COND record containing temperature information. Both these pieces of information may be in the same COND record. There may be any number of COND records containing traffic and construction information.

a. This type of transaction does not have TITEM or IDISC or TTAX or TTEND records.

b. Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Reference Number Column	Meaning	Req?
1	Weather – code type 'WEAT'	N
2	Temperature – a signed 3 digit number.	N
3	Traffic – code_type 'TRAF'	N
4	Construction – code_type 'CONS'	N

11. Transaction of type 'TOTAL':

This transaction records the totals that are reported by the POS. The value field must be populated. Some POS systems generate only one transaction number for all totals. In order to avoid duplicate errors to be reported, only one total transaction can have a transaction number and the subsequent ones can have blank transaction numbers. In other words, a TOTAL transaction is not required to have a transaction number.

This type of transaction does not have TITEM or IDISC or TTAX or TTEND records.

12. Transaction of type 'METER':

This transaction is generated when a meter reading of a fuel pump is taken.

- a. This type of transaction has only TITEM records.
- b. Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Reference Number Column	Meaning	Req?
1	Reading Type: ('A' Adjustment, 'S' shift change, 'P' price change, 'C' store close)	Y
5	Opening Meter Readings	Y
6	Closing Meter Reading	Y
7	If the reading type is 'P' for price change, the old unit retail should be placed here. Decimal places are required.	Y
8	Closing Meter Value	Y

13. Transaction of type 'PUMPT':

This transaction is generated when a pump test is performed. This type of transaction has only TITEM records.

14. Transactions of type 'TANKDP':

This transaction is generated when a tank dip measurement is taken.

- a. This type of transaction has only TITEM records.
- b. Meaning of reference number fields:

Note: The meaning of these reference number fields may be changed through the sa_reference table.

Reference Number Column	Meaning	Req?
1	Tank identifier	Y
5	Dip Type ('FUEL', 'WATER', etc.)	Y
6	Dip Height Major (decimal places required)	Y
7	Dip Height Minor (decimal places required)	Y

15. Transaction of type 'DCLOSE':

This transaction is generated when day closed. Transaction number for this type of transaction has to be blank.

16. A note about vouchers: Vouchers are minimally handled by saimptlog. Voucher information is written to the savouch file which is passed to the program savouch.pc. For more information about this interface, see Interface File – SA Vouch and Batch Design – savouch.

A voucher will appear on the TITEM record only if it was sold. Thus when saimptlog encounters a 'SALE' transaction with a voucher, it writes the voucher to the savouch file as an 'Issued voucher'.

A voucher will be issued when it appears on the TTEND record of transactions of type 'RETURN' and 'PAIDOU'. In other words, saimptlog will write it to the savouch file with status 'I'.

A voucher will be redeemed when it appears on the TTEND record of transactions of type 'SALE' and 'PAIDIN'. In other words, saimptlog will write it to the savouch file with status 'R'.

Vouchers may not be returned. However, a transaction of type 'PAIDOU' may be generated when the customer exchanges a voucher for another form of tender.

Sales Audit Interface to Oracle 10.7'

Column Name	Column Type	Default Value	Description
Status	Varchar2(50)	NOT NULL	This column represents the type of posting being applied. All inserts from this module, status should be set to 'NEW'
Set_of_books_id	Number(15)	NOT NULL	This column represents the identifier for the book of accounts that this module will be posting to. This field should always be set to the value found in FIF_ORA_GL_SETUP.SET_BOOKS_ID
User_je_source_name	Varchar2(25)	NOT NULL	Journal entry source name for the posted transaction. This entry must exist in the Oracle USER_JE_SOURCE_NAME column in the Journal Sources table prior to posting data to the GL. This value should be hard-coded to 'ReSA'
User_je_category_name	Varchar2(25)	NOT NULL	Journal entry source name for the posted transaction. This entry must exist in the Oracle USER_JE_CATEGORY_NAME column in the Journal Categories table prior to posting data to the GL. This value should be hard-coded to 'ReSA'.
Accounting_date	Date	NOT NULL	The date of the transaction/total – SA_STORE_DAY.BUSINESS_DATE
Currency_code	Varchar2(15)	NOT NULL	The default system currency code
Date_created	Date	NOT NULL	period.vdate

Column Name	Column Type	Default Value	Description
Created_by	Number(15)	NOT NULL	This field represents the identifier of the application/user whom created this journal entry. This value should be populated with the FIF_ORA_GL_SETUP.LAST_UPDATED_ID.
Actual_flag	Varchar2(1)	NOT NULL	The hard-coded value 'A' will represent actual amounts
Encumbrance_type_id	Number	NULL	
Budget_version_id	Number	NULL	
Currency_conversion_date	Date	NULL	The date in which the total was converted to the default currency code. This value should be populated with the store day bussiness date
User_currency_conversion_type	Varchar2(30)	NULL	This value should be hard-coded to 'Spot'
Currency_conversion_rate	Number	NULL	
Segment1 – 30	Varcharr(25)	NULL	These columns should be populated with either the debit segment values or the credit values (depending on which half of the total you are posting).
Entered_dr	Number	NULL	If you are entering the debit half of the total, place the total amount in this column. If you are representing the credit half of the total, place a 0 in this column.
Entered_cr	Number	NULL	If you are entering the credit half of the total, place the total amount in this column. If you are representing the debit half of the total, place a 0 in this column
Accounted_dr	Number	NULL	
Accounted_cr	Number	NULL	

Column Name	Column Type	Default Value	Description
Transaction_date	Date	NULL	
Reference1 – 30	Varchar2(100)	NULL	These columns will be hard-coded to NULL for the total exports
Group_id	Number(15)	NULL	
Je_batch_id	Number(15)	NULL	
Period_name	Varchar2(15)	NULL	
Je_header_id	Number(15)	NULL	
Je_line_num	Number(15)	NULL	
Chart_of_accounts_id	Number(15)	NULL	
Functional_currency_code	Varchar2(15)	NULL	
Code_combination_id	Number(15)	NULL	If this is the debit half of the total adjustment, place the FIF_ORA_CROSS_REF.D B_CCID. If this is the credit half of the total adjustment, place the FIF_ORA_CROSS_REF.C R_CCID
Date_created_in_GL	Date	NULL	
Warning_code	Varchar2(4)	NULL	
Status_description	Varchar2(240)	NULL	
Desc_flex_error_message	Varchar2(240)	NULL	
Stat_amount	Number	NULL	
Request_id	Number(15)	NULL	
Subledger_doc_sequence_id	Number	NULL	
Subledger_doc_sequence_value	Number	NULL	
Ussgl_transaction_code	Varchar2(30)	NULL	
Attribute1 –20	Varchar2(150)	NULL	
Context1-3	Varchar2(150)	NULL	
Invoice_date	Date	NULL	
Invoice_amount	Number	NULL	
Invoice_identifier	Varchar2(20)	NULL	
Tax_code	Varchar2(15)	NULL	

‘Metered Sales Volumes’

The information for this API is computed by the following total functions XXXX, and placed on the sa_total table.

Column Name	Column Type	Values	Database Column	Description
Store	Varchar2(4)	Specified by external system	store.store	Store identifier
Trading Period Date	Date	Transaction Date and Time	sa_total.ref_no1	Date and time the last transaction in this group occurred on the POS (YYYYMMDDHH24MISS).
Business Day	Varchar2(8)	Business Date	sa_store_day.business_date	Business date of transactions (YYYYMMDD).
Pump	Varchar2(8)	Specified by external system	sa_tran_item.ref_no2	Pump id of gas sales. The sa_tran_item.ref_no2 field is in two parts, separated by a :. Pump id is the part of the field before the :.
Product	Varchar2(15)	Item number of the fuel sold	sa_tran_item.ref_no2	SKU number. It is the part of the field after the :.
Unit Price	Number(20,4)	Price of item.	sa_tran_item.ref_no3	Unit retail.
Sold Volume	Number(15,4)		Qty = value/unit_retail	Actual quantity of this grade sold at this store/date/pump/price.
Non-sold Volume	Number(15,4)		Qty = value/unit_retail	Quantity lost through drive-offs.
Pump Test Volume	Number(15,4)		Qty = value/unit_retail	Quantity of fuel dispersed for pump testing.
Sold Value	Number(20,4)		sa_total.total_value for total_id of usage_type ‘SFMS’	Retail value of ‘sold’ volume.

Column Name	Column Type	Values	Database Column	Description
Non-sold Value	Number(20,4)		sa_total.total_value for total_id of usage_type 'SFMD'	Retail value of 'non-sold' volume.
Pump Test Value	Number(20,4)		sa_total.total_value for total_id of usage_type 'SFMP'	Retail value of 'pump test out' volume.
Trading Period Number	Varchar2(4)	01, 02, 03	Null	Shift number.
Currency Code	Varchar2(15)		store.currency_code	Currency used at store.
Back Office Ref	Varchar2(41)	total_seq_nos	sa_total.total_seq_no: sa_total.total_seq_no	Total sequence number of sales total and total sequence number of non-sales total concatenated with a ":" in between.

Meter Reading

Column Name	Column Type	Default Value	Database Column	Description
Store	Varchar2(4)	Specified by external system	store.store	Store identifier
Trading Period	Date	Transaction Date and Time	sa_tran_head.trans_date	Date and time the transaction occurred on the POS (YYYYMMDDHHMISS)
Business Day	Varchar2(8)	Business Date	sa_store_day.business_date	Business date of transactions (YYYYMMDD).
Pump	Varchar2(8)	Specified by external system	sa_tran_item.pump	Pump id number of gas sales
Product	Varchar2(15)	Item number of the fuel sold	sa_tran_item.sku	SKU number

Column Name	Column Type	Default Value	Database Column	Description
Unit Price	Number(20,4)		sa_tran_item.unit_retail	Unit retail.
Reading Type	Varchar2(6)	Close, Price change, Adjustment, Shift Close	sa_tran_head.ref_no1	Type of reading taken.
Opening Reading	Number(12,1)		sa_tran_item.ref_no5	Opening reading
Closing Reading	Number(12,1)		sa_tran_item.ref_no6	Closing reading
Opening Value	Number(12,1)		sa_tran_item.ref_no7	Opening value
Closing Value	Number(12,1)		sa_tran_item.ref_no8	Closing value
Trading Period Number	Varchar2(6)	01, 02, 03	Null	Shift number.
Comments	Varchar2(240)		sa_tran_head.comments	Text to allow for explanation for a reset.
Back Office Ref	Varchar2(20)		sa_tran_head.trans_seq_no	transaction sequence number

Tank Dip

Column Name	Column Type	Default Value	Database Column	Description
Store	Varchar2(4)	Specified by external system	store.store	Store identifier
Business Day	Varchar2(8)	Business Date	sa_store_day.business_date	Business date of transactions (YYYYMMDD).
Tank	Varchar2(8)	Specified by external system	sa_tran_head.ref_no1	Tank id for this dip.
Product	Varchar2(15)	Item number of the fuel sold	sa_tran_item.sku	SKU number
Tank Dip Closing Volume	Number(15,4)		sa_tran_item.qty	There are two sa_tran_item records for this transaction. sa_tran_item.ref_no 5 tells us whether
Tank Dip Height Major	Number(5)		sa_tran_item.ref_no6	

Column Name	Column Type	Default Value	Database Column	Description
Tank Height Minor	Number(5)		sa_tran_item.ref_no7	5 tells us whether the fields are for the FUEL fields or for the WATER fields.
Water Closing Volume	Number(15,4)		sa_tran_item.qty	
Water Height Major	Number(5)		sa_tran_item.ref_no6	
Water Height Minor	Number(5)		sa_tran_item.ref_no7	
Back Office Ref	Varchar2(20)		sa_tran_head.tra_seq_no	transaction sequence number
Comments	Varchar2(240)		sa_tran_head.comments	Comments about the dip

File layouts input and output

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by saimptlog/external system	Line number of the current line of the file
	File Type Definition	Char(4)	SAVO	Identifies file as 'Savoucher'
	File Create Date	Date (14)	Create date	Date and time file was written by saimptlog or external system (YYYYMMDDHHMMSS)
	Business Date	Date(8)	business date to process	saimptlog Business date of transactions (YYYYMMDD)
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	Specified by saimptlog/external system	ID of the current line being processed by input file
	voucher_seq_no	Number(20)	Specified by saimptlog/external system	Unique voucher identifier.

Record Name	Field Name	Field Type	Default Value	Description
	Voucher_no	Char(16)	Specified by saimptlog/external system	Voucher number.
	Tender_type_id	Number(6)	specified by saimptlog/external system	The tender type of the voucher.
	Ass_date	Date(8)	specified by saimptlog/external system	Date the voucher was assigned(YYYYM MDD)
	Ass_store	Number(4)	specified by saimptlog/external system	Store the voucher was assigned to.
	Iss_date	Date(8)	specified by saimptlog/external system	Date the voucher was issued/sold. (YYYYMMDD)
	Iss_store	Number(4)	specified by saimptlog/external system	Store that issued/sold the voucher.
	Iss_register	Char(5)	specified by saimptlog/external system	Register that issued/sold the voucher.
	Iss_cashier	Char(10)	specified by saimptlog/external system	Cashier that issued/sold the voucher.
	Iss_tran_seq_no	Number(20)	specified by saimptlog/external system	Transaction number of issue/sale.
	Iss_item_seq_no	Number(20)	specified by saimptlog/external system	Sequence of the item in the transaction.
	Iss_tender_seq_no	Number(20)	specified by saimptlog/external system	Sequence of the tender in the transaction.
	Iss_amt	Number(20,4)	specified by saimptlog/external system	Amount of the issue/sale.
	Iss_cust_name	Char(40)	specified by saimptlog/external system	The name of the customer who was issued the voucher.
	Iss_cust_addr1	Char(40)	specified by saimptlog/external system	The address of the customer who was issued the voucher.

Record Name	Field Name	Field Type	Default Value	Description
	Iss_cust_addr2	Char(40)	specified by saimptlog/external system	The second address line of the customer who was issued the voucher.
	Iss_cust_city	Char(30)	specified by saimptlog/external system	The city of the customer who was issued the voucher.
	Iss_cust_state	Char(3)	specified by saimptlog/external system	The state of the customer who was issued the voucher.
	Iss_cust_postal_code	Char(10)	specified by saimptlog/external system	The postal code of the customer who was issued the voucher.
	Iss_cust_country	Char(3)	specified by saimptlog/external system	The country of the customer who was issued the voucher.
	Recipient_name	Char(40)	specified by saimptlog/external system	The name of the intended recipient of the gift certificate.
	Recipient_state	Char(3)	specified by saimptlog/external system	The state of the intended recipient of the gift certificate.
	Recipient_country	Char(3)	specified by saimptlog/external system	The country of the intended recipient of the gift certificate.
	Red_date	Date(8)	specified by saimptlog/external system	Date of redemption. (YYYYMMDD)
	Red_store	Number(4)	specified by saimptlog/external system	Store of redemption.
	Red_register	Char(5)	specified by saimptlog/external system	Register of redemption.
	Red_cashier	Char(10)	specified by saimptlog/external system	Cashier of redemption.
	Red_tran_seq_no	Number(20)	specified by saimptlog/external system	Transaction number of redemption.

Record Name	Field Name	Field Type	Default Value	Description
	Red_tender_tran_no	Number(20)	specified by saimptlog/external system	Sequence of tender in transaction where voucher is redeemed.
	Red_amt	Number(20,4)	specified by saimptlog/external system	Amount of redemption.
	Exp_date	Date(8)	specified by saimptlog/external system	Voucher expiration date. (YYYYMMDD)
	Status	Char(1)	Specified by saimptlog/external system	Voucher status - can be either 'SAVS_A'-assigned, SAVS_I-issued, SAVS_R - redeemed.
	Comments	Char(255)	specified by saimptlog/external system	Any comments associated with the voucher.
	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
File Trailer	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Record Counter	Number(10)		Number of records processed in current file (only details records between FHEAD and FTAIL)

Reject File

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will need to be created by the voucher upload module and a reject line counter will be required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and trailer record and the file line count will be equal to 0.

The reject filename should also be specified as a runtime parameter.

Error File

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance.

Chapter 15 - Batch Functional Overviews

Allocations

Functional Overview

The Allocation Update Download (allocupd) module will download new item retail price and ticketing information for allocations that have been previously downloaded to a warehouse management system. This will allow the warehouse to correctly ticket the items cross docked to store locations.

Module Flow and Scheduling

The Allocation Update (allocupd) module should be run after the purchase order and allocations have been downloaded (ie. ordrev) and after the pricing batch modules in order to send the most current retail to the warehouse.

The following diagram 1 outlines the individual modules that are associated with the Allocations dialog:

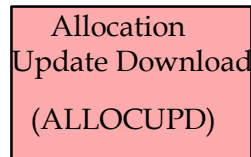


Diagram 1

Audit Trail

Functional Overview

The Audit Trail batch component performs two major functions:

- Activates or deactivates the audit trail functionality for the appropriate tables
- Purges old information from the audit tables according to a predetermined schedule.

The Audit Logic Information Edits (auditsys) module performs the activation and deactivation of the audit functionality. You must first select a RMS table (the driver table) for auditing online. When executed, this program dynamically creates the tables that hold audit information for the specified driver table. Once the audit table exists, a record will be inserted into it whenever a record on the driver table is inserted, updated, or deleted. Conversely, this program will dynamically delete the audit tables for any driver table that has been deactivated online.

The Audit Purge Process (auditprg) module deletes records from audit tables according to the frequency set online for each RMS table being audited. The audit table purge frequency can be set to 'Daily', 'Weekly', 'Monthly', 'Semi-Annually', or 'Yearly'.

Module Flow and Scheduling

The Audit Logic Information Edits (auditsys) module is designed to run daily; however, it really only needs to be executed when new audit information has been requested. The Audit Purge Process (auditprg) module can be scheduled to run at any time; however, practical usage is to have it run only once at the end of each month when all batch cycles are finished.

The following diagrams outline the programs that are associated with the Audit Trail dialog:

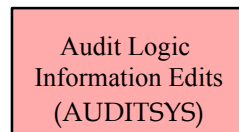


Diagram 1

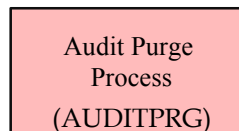


Diagram 2

Bill of Lading

Functional Overview

The only batch component in the Bill of Lading dialog is the Bill of Lading Purge module. The function of this module is to delete old bill of lading files that have exceeded the retention period. The retention period is maintained online in the System Options dialog, by the system administrator.

Module Flow and Scheduling

The Bill of Lading Purge (ladprg) module can be scheduled to run at any time; however, practical usage is to have it run only once at the end of each month near the end of the batch cycle.

The following diagram 1 outlines the program that is associated with the Bill of Lading dialog:

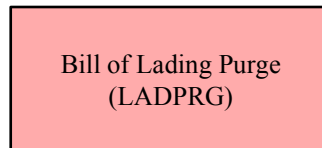


Diagram 1

Calendar

Functional Overview

The Calendar batch component updates the system date used for all processing runs, reports, and module execution. The user has the option of specifying a new system date. If one is not specified, the module will add one day to the current system date.

Module Flow and Scheduling

The Increment Set System Date (dtesys) module should be run daily, weekly and monthly to update both the end-of-week and end-of-month dates.

This program should be the very last one run in the batch cycle.

The following diagram 1 outlines the individual module that is associated with the Calendar dialog:

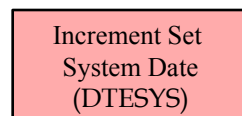


Diagram 1

Clearance

Functional Overview

The Clearance function is designed to provide an orderly and efficient framework for the authorization and control of clearance markdowns. In order to do this, the RMS uses the concept of *events*, similar to the events used in Pricing. Clearance events allow multiple markdowns of the same item/zone. Also, the Clearance price for the item is not permanent—the retail price will be reset to the regular retail after the reset date specified for the event has passed. Therefore the batch processes that run in the Clearance function of the RMS have two distinct parts – processing markdown prices and processing reset prices.

The Clearance Pricing Download (pccdnld) module is run in Phase 1 of the batch schedule to send clearance markdown retail prices to the point of sales system. Clearance markdowns that are to take effect within the predetermined number of days are written out to a table that will be used as an interface point with the point of sale and the clearance detail records are updated with the current date as a downloaded date. The number of days prior to taking effect that clearance markdowns are downloaded is maintained as a system option.

The Retek Clearance Pricing Extract (pccext) module is run in Phase 3 of the batch schedule to extract the new clearance event retail pricing from the clearance event tables and update the retail information in RMS. Clearance markdowns that are to take place on the following business day are selected and the appropriate SKU/store tables have their retail prices update to the current clearance markdown prices and their clearance indicators updated to Yes.

The Reset Pricing Download (pccrdnld) module is also run in Phase 1 of the batch schedule to send clearance reset pricing information to the point of sales. When a clearance event is within a predetermined number of days of its reset date, this program will gather current clearance pricing information and the pricing information to which an item will be reset. This information will be written out to a table that will be used as an interface point with the point of sale. Clearance records will be updated to indicate that they have had reset prices downloaded to the point of sale by setting a reset downloaded date with the current date.

The Retek Clearance Reset Price Extract (pccrext)also runs in Phase 3 of the batch schedule to update the SKU/store tables with the regular retail prices and to change the clearance indicator back to No. The item/locations are selected from the clearance tables the night before the pricing reset is to take place.

The Clearance Event Deletions (pccprg) module removes completed, canceled, and rejected clearance events from the system. Before a clearance event can be deleted, a pre-determined number of months must have passed since the reset date of the clearance. The pre-determined number of months is based on the value entered for the system option Clearance Retention Months.

Module Flow and Scheduling

The Clearance Pricing Download (pccdnlld) module is run on a daily basis during Phase 1 of the batch schedule. The Reset Pricing Download (pccrdnlld) module is also run during Phase 1 but only after the Price Change POS Extract (pcdnld) module from the Pricing dialogue has terminated successfully.

The Retek Clearance Pricing Extract (pccext) module followed by Retek Clearance Reset Price Extract (pccrext) are run daily during Phase 3 of the batch schedule only after the price change transaction (pctran) module from the Pricing dialogue has terminated successfully.

The Clearance Event Deletions (pccprg) module can be scheduled to run at any time; however, practical usage is to run it once a day at the end of all batch cycles.

The following diagram 1 outlines the individual programs that are associated with the Clearance dialog:

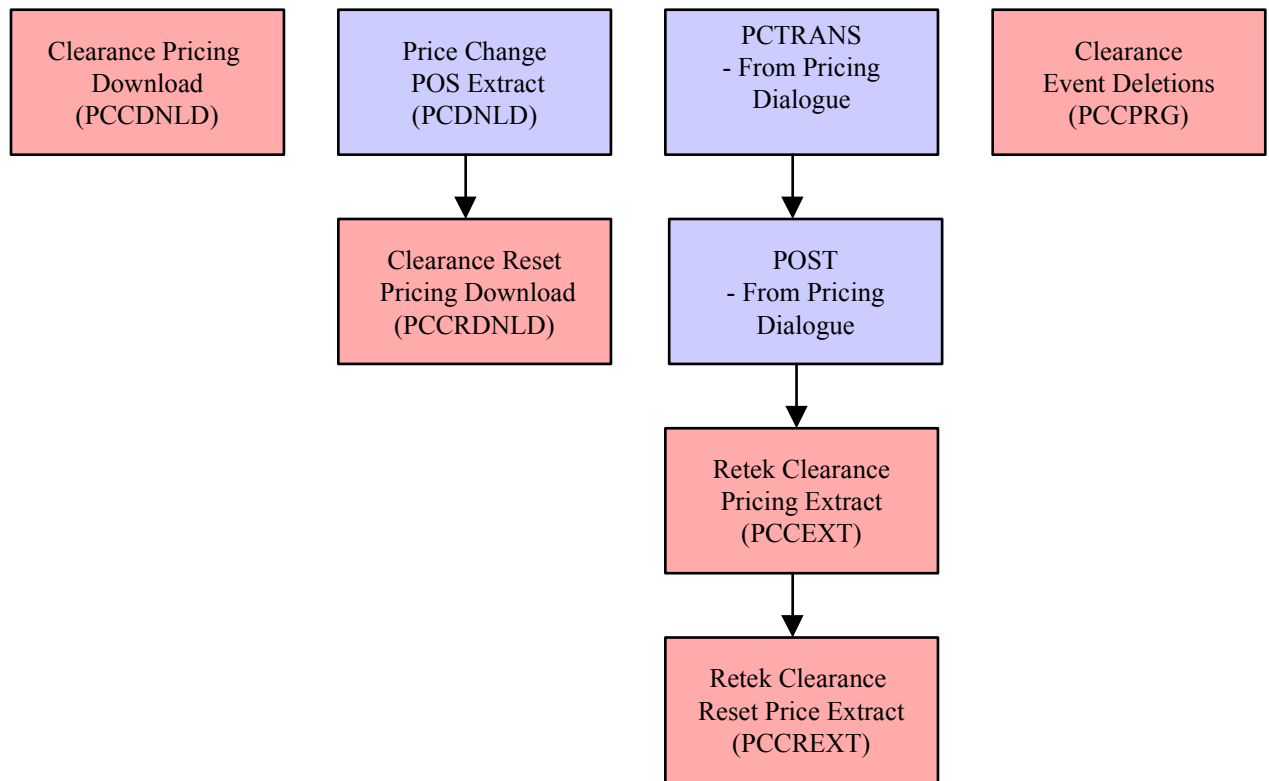


Diagram 1

Color and Size Dimensions

Functional Overview

The Color and Size Dimensions batch component is used to build the size ratios for all size combinations previously set up online. The batch component consists of the Size Ratio Build (szrtbld) module. This module creates size ratios for all size combinations that exist for the department, class, or subclass by using the size ratio IDs set up online. These ratios are also set up at the subclass or store level. Only size ratios that are due to be rebuilt or have been recently updated online are processed by Size Ratio Build. The size ratios are written to a file, which are loaded into the Retek System tables when the Retek RMS system is not under heavy use.

Module Flow and Scheduling

The Size Ratio Build (szrtbld) module should be run on a daily basis.

This program will likely be run after sales information is uploaded into Retek.

The following diagram outlines the individual programs that are associated with the Color and Size Dimensions dialog:

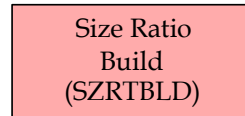


Diagram 1

Complex Deals Maintenance

Functional Overview

The Complex Deals Maintenance batch component is used to calculate order discounts and rebates, calculate estimates of net cost, net net cost and dead net net cost for items with deals applicable to them, and to close deals and purge deal data.

The Deal Item Insert (ditinsrt) module explodes records on the DEAL_QUEUE table onto the DEAL_CALC_QUEUE and DEAL_SKU_TEMP tables. DEAL_QUEUE holds the IDs of any deals that have been modified during the day, and thus must have cost estimates and orders updated accordingly.

The Order Discount Calculation (orddscent) module calculates discounts and rebates applicable to purchase orders. The module runs based on all orders inserted into the DEAL_CALC_QUEUE table. Orders are inserted into DEAL_CALC_QUEUE either when the order is modified (approved or deal calculation is requested on-line), or when a deal associated with the order is modified.

The Deal Cost Calculation (dealcalc) module calculates estimates of net cost, net net cost and dead net net cost for items with deals applicable to them.

The Deal Close (dealcls) module closes any active deals that have reached their close date.

The Deal Purge (dealprg) module purges deals that are in closed status and have been in the system a system-options-specified number of days.

Module Flow and Scheduling

The Deal Item Insert (ditinsrt) module should be scheduled to run daily, during Phase 1 of the batch process. It should run before orddscent (nightly run) and dealcalc.

The Order Discount Calculation (orddscent) module can be run ad hoc throughout the day, so that users can have access to updated order information upon request. It should also be scheduled to run daily, during Phase 2 of the batch process. It should run after ditinsrt, rclsdlly and sccext, and before edidlrd.

The Deal Cost Calculation (dealcalc) module should be scheduled to run daily, during Phase 2 of the batch process. It should run after ditinsrt.

The Deal Close (dealcls) module should be scheduled to run daily, during Phase 3 of the batch process.

The Deal Purge (dealprg) module can be scheduled ad hoc, whenever deal information needs to be purged from the system. It should be run at least monthly.

Diagram 1 outlines the programs that are associated with the Complex Deals Maintenance dialog:

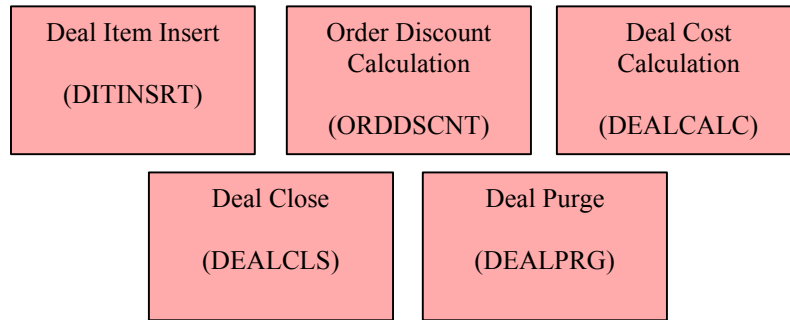


Diagram 1

Contracts

Functional Overview

The Contracts batch component is executed in order to create purchase orders from contracts and purge obsolete contracts.

The Contract Maintenance and Purging (cntrmain) module purges contracts that have remained in Cancelled, Worksheet, or Submitted status for a user-defined number of months. Additionally, the status will be reset for active contracts that have been inactive (have not been ordered against) for a user-defined number of months. These time periods are maintained in the system administration dialog.

The Contract Replenishment – Type B Contracts (cntrordb) module creates purchase orders for all type B contract items that are ready to have orders raised against them. Contract, SKU, and location information is selected from the contracting tables where production dates are ready to be met. An order is written for each contract and all of the items and locations on the contract. It should only be run if the system setting for contracting is set and type B contracts are used. The Replenishment Order Maintenance module will then run and evaluate the orders created by the Contract Replenishment – Type B Contracts module.

The Contract A/C/D Replenishment Processing (cntrprss) module evaluates contracts of type A, C, and D. Contracts are ranked so that orders are called off of the best contracts first. The criteria for ranking are lead-time (earliest is best), cost (cheapest is best), contract status (closed preferred over open), and contract type (type C are preferred over D). It updates the temporary orders created by the Item Replenishment Extract (rplext) module with the contract and supplier information of the best available contract for each item. If the replenishment need for all item/locations is greater than the availability on the best contract, then the best supplier will fulfill as many requirements as possible and then new temporary order records will be created using contracted amounts from the next best contracts available. If the need of all item/locations is greater than the total availability for all of the contracts for that item, then a rationing algorithm will apportion the available stock to item/locations according to greatest need. If the system requires that all orders be called from contracts, then any unfulfilled amount will be written to a table that tracks replenishment needs that have to be met.

The EDI Contract module downloads a file of EDI contract information, processing contracts only if they are in Approved status and are EDI contracts.

Module Flow and Scheduling

The Contract Maintenance and Purging (cntrmain) module and EDI Contract (edidlcon) module should be run on a daily basis.

The Contract Replenishment – Type B Contracts (cntrordb) module should be run after the Contract Maintenance and Purging (cntrmain) module and EDI Contract (edidlcon) module and before the Replenishment Order Maintenance (rplprg) module.

The Contract A/C/D Replenishment Processing (cntrprss) module should be run after the Replenishment Extract (rplext) module and before the Replenishment Order Build (rplbld) module.

The following diagram 1 outlines the individual programs that are associated with the Contracts dialog:

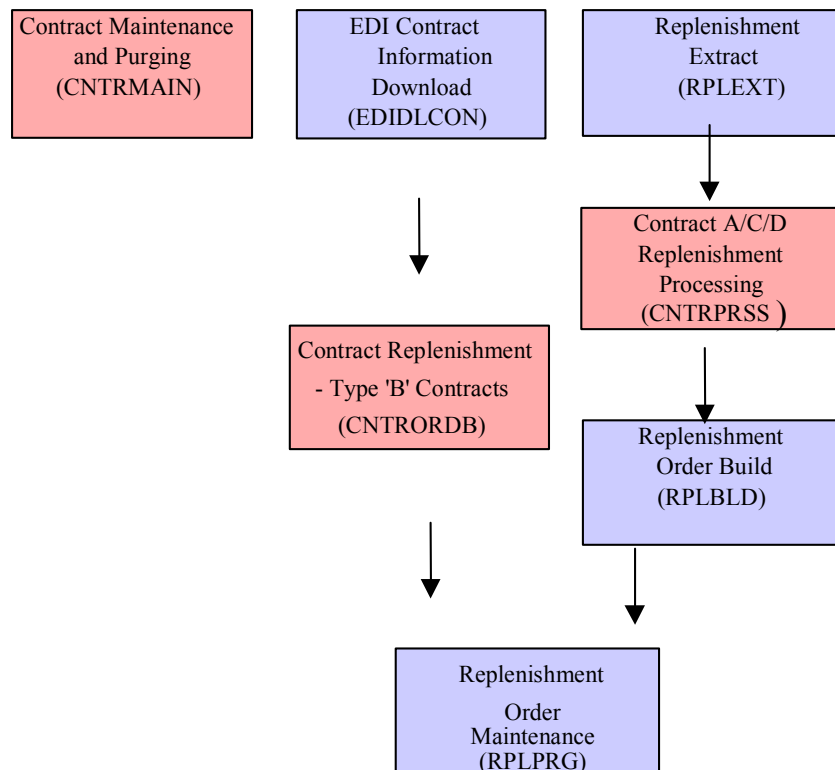


Diagram 1

Customs Entry (CE)

Functional Overview

The CE batch component is executed in order to send Customs Entry information to a given Broker for review and correction.

Module Flow and Scheduling

The processing of Customs Agencies' information (cednld) should be run in the nightly batch schedule.

The following diagram 1 outlines the individual modules that are associated with the Customs Entry dialog:

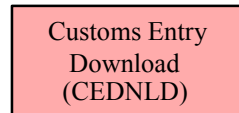


Diagram 1

EDI

Functional Overview

The EDI batch component is executed in order for a user of the RMS to communicate item, ordering, pricing, sales, and receiving information to and from vendors.

The Supplier Availability for Contract Upload (ediupavl) module is run to upload a supplier availability schedule, or what items the suppliers have available. This is associated with the contracts functionality. For more information see the contracts dialog.

The EDI Contract Information Download (edidlcon) module downloads contract information into a file. Contracts are only processed if they have been approved and the user has checked the EDI Contract check box on the Contract Header Detail form in RMS.

The Implementation of the ASN Upload (ediupasn) module is run to upload advanced shipment notification records, including what orders and items are contained within those shipments. These created records are used to create shipments within RMS.

The New and Changed Item Upload form Supplier (ediupcat) module is run to upload new and changed item information, and item cost change information from suppliers. By updating this information, users can to view and implement vendor changes in RMS.

The New and Changed Supplier Address Upload (ediupadd) module is run to upload supplier address change information. These files will be used to update RMS supplier address information.

The Supplier Order Acknowledgements and Changes (ediupack) module is run to give EDI notification of vendor generated orders. The program acknowledges vendor receipt of a buyer generated order without changes; acknowledges vendor receipt of a buyer generated order with date, cost, or quantity modifications; notifies a buyer of a vendor-generated order; and acknowledges order cancellations. As a result of this program being run, a record containing the ordering information is updated or created.

The Retek Store and Warehouse Address Download (edidladd) module downloads store and warehouse address lists for vendors. If the user has checked the Create EDI Address Catalog check box on the System Control form in RMS, the addresses of all stores and warehouses will be downloaded. Otherwise, only addresses that have been changed will be downloaded.

The Sales and Stock Activity Report Download (edidlprd) module is run to give EDI notification to vendors or daily and weekly sales audit summaries. The summaries include sales details, current stock on hand for all locations, and current in transit quantities for each of the SKUs primarily supplied by each vendor.

The New and Changed P.O. Download (edidlrd) module is run to write orders generated within the RMS to a file if they have been approved and the user has checked the EDI PO check box on the Order Header Detail form. This module handles both new and changed orders. The most-recent order and allocation information will be used, to ensure that the latest changes are being sent.

The EDI purge (ediprg) module will purge rejected EDI new items or cost changes. New or changed item information and cost change information sent by suppliers will be able to be approved or rejected online. Existing items that are rejected will be removed from the system when they exist longer than the number of days specified in RMS (the number of days are set in the system administration dialog).

Module Flow and Scheduling

The ASN Upload (ediupasn) module, EDI Contract Information download (edidlcon) module, the Sales and Stock Activity Report Download (edidlprd) module, and New and Changed P.O. Download (edidlrd) module should be scheduled to run on a daily basis after the ordering functionality has completed. The order of execution is explained above.

The Supplier Availability for Contracts Upload (ediupavl), New and Changed Item Upload from Supplier (ediupcat) module, New and Changed Supplier Address Upload (ediupadd) module, Supplier Order Acknowledgements and Changes (ediupack) module, the Retek Store and Warehouse Address Download (edidladd) module can be scheduled to run at any time, but the recommended order of execution is outlined above.

The EDI Purge (ediprg) module can be scheduled to run at any time. However, practical usage is to run it only once at the end of each month at the end of all batch cycles.

The following diagram 1 outlines the individual programs that are associated with the EDI dialog:

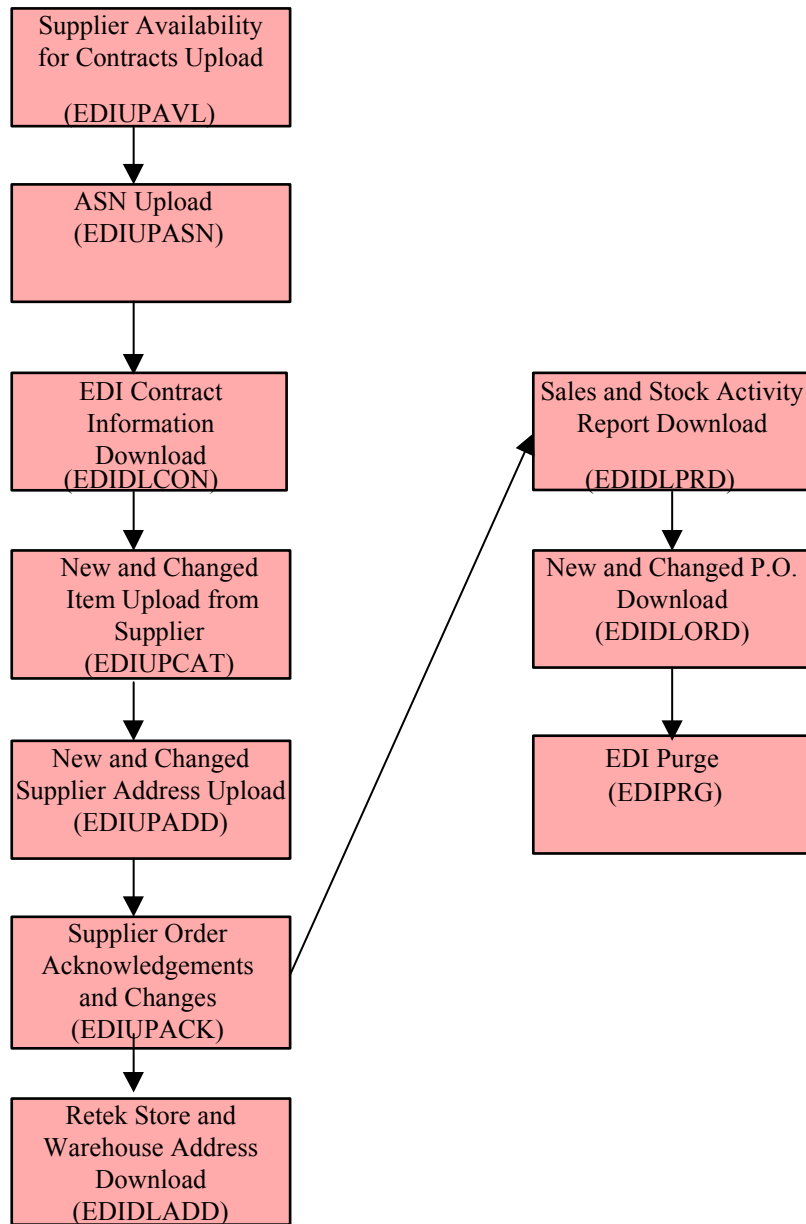


Diagram 1

Functional Constraints and Limitations

As can be expected with two different systems, the Oracle Financial Application and the Retek Merchandising System may be different in some ways in certain areas. For example, the applications may contain different table structures. This section describes some constraints that users should be aware of.

Vendors

1. Oracle po_vendors.segment1 will be used as the supplier number on the RMS side. To this end, it should only receive unique numeric values 10 digits or less.
2. To further support point 1, segment1 should be initialized to 1 when the system is installed. If automatic sequence1 generation is selected, this will provide the maximum number of vendors before the RMS's supplier ID numbers become out of range (> 9999999999).
3. The RMS only supports one contact per address. Only one contact should be entered per address on the Oracle side.
4. The vendor bridging process uses partitioned views to retrieve its data. Users that will be driving the staging of vendor information (either via the batch process or the RMS conversation) must have the correct organization ID in their personal schema, or the views will not retrieve the correct data (in some cases, any data).
5. The RMS will use the EDI indicator located on the flexfields pane. Regardless of action taken on the EDI window, this yes/no field should be set correctly. The RMS does not have a separate field for provinces. This information should be handled in the Oracle State field (and have correct codes on the state information tables set up).
6. The Interface is capable of pulling only those suppliers created in Oracle Financials whose Supplier Types includes any one of them defined in the RMS Code_detail table. The following command will display all the Supplier Types defined in RMS ;

```
Select * from code_detail where code_type = 'SUTY';
```

One has to create the Supplier Types also in Oracle Financials through Quick Codes. The path to create this in Oracle Financials is as follows:

Oracle Payables -> Setup -> QuickCodes -> Payables

Then Type = 'Vendor Type' and one has to add appropriate name to the

Name field which should match with code field defined in code_detail table for the code_type = 'SUTY'.

The Suppliers created should include those Payments and Freight Terms which were pulled from Oracle Financials and exists in RMS Terms and Freight Terms table. The Supplier Type should also exists in the code_detail table for the code_type = 'SUTY'.

The Supplier information is brought over to the RMS sups table and the Partner information to the RMS partner table. The address information is captured in the addr table and to view the Supplier and Partner addresses, one has to select the key_value_1 and key_value_2 for suppliers and partners respectively.

If there is any updates to the address lines1, address lines2 and address lines3, the interface pulls the updated informations.

Payment Terms

Oracle's term-name will be used as the RMS's term identifier. RMS supports upto 15 characters for Terms .It is suggested that Payment Terms should not be edited in Oracle Financials for historical retention of data and would be advisable to create a new term. The Interface picks only the first line from the Payment Terms Lines in Oracle Financials. The effective dates for terms are not handled within the RMS side.Any updates to the Payment Lines Types will be brought over to RMS for the RMS9.0 – OF10.7 release.

Freight Terms

1. Oracle's Lookup Codes will be the RMS's freight term identifier. The RMS only supports a freight term ID of 2 characters. Oracle users should not enter non-numeric displayed field lengths longer than 2 characters.
2. It is suggested that FreightTerms should not be edited in Oracle Financials for historical retention of data and would be advisable to create a new term. Any updates to the description field in Oracle Financials can be passed to RMS.

Currencies

1. It is assumed that when the systems are initially set up, identical currency information (3 letter codes, exchange rate values, etc.) will be input to both the RMS and Oracle. If a new currency should become necessary, it must be entered into both Oracle and the RMS before making a rate change (or being used in any other way, such as on a purchase order). There will be no functionality for bridging these data.
2. There is no notification/export/staging table used on Oracle's side. The loading process simply goes through the rates table and selects rate information that is VDATE + 1. Because of this, after VDATE is rolled forward, there is no way to go back and get rates that didn't stage correctly. To combat this as much as possible, the interface errors table should be checked for rates that did not successfully stage after the loading process runs. It can be run again before VDATE is rolled and get the remainder of the appropriate rates.

Purchasing/Receiving

1. A TRANSACTION_TYPE is sent with every transaction. Valid values are:

CREATE

UPDATE

CANCEL

2. Certain Attribute fields must always be populated. Attribute fields will be populated with our equivalent of a primary key.

PO_HEADERS – Attribute1 will be populated with ordhead.order_no.
Segment1 will also be populated with ordhead.order_no.

PO_LINES – Attribute1 will be populated with ordhead.order_no, Attribute2 will be populated with ordsku.sku

PO_LINE_LOCATIONS – Attribute1 will be populated with ordhead.order_no, Attribute2 will be populated with ordsku.sku. Attribute3 will be populated with ordloc.location

PO_DISTRIBUTIONS – Attribute1 will be populated with ordhead.order_no, Attribute2 will be populated with ordsku.sku. Attribute3 will be populated with ordloc.location. Attribute 4 will be populated with the account that will be charged the amount for an order/location, that will be ordloc.location.

3. Oracle has agent_id on the headers table. Agent_id will be defaulted to the RMS user. This RMS user will be created as an employee field on the HR_EMPLOYEES table.

4. The last_updated_by and created_by will be defaulted to the RMS user. As a result, a userid must be set up on Oracle as 'Retek'.
5. Only orders that have types of Approved, Cancelled or Received will be bridged to Oracle.
6. Table: PO_LINES_INTERFACE_CAI. The unit_meas_lookup_code – Oracle must have the RMS codes on their mtl_units_of_measure table. The field name is unit_of_measure. Our codes on code detail are 'PRUM' — eaches and case ('EA' and 'CA'). Since the RMS currently only allows the usage of 'EA', one uom_code must be created.

RMS Chart of Accounts Interface to Oracle Financials

This process bridges new or updated RMS Stock Ledger mapping fields (department, class, subclass, location, transaction code and Line type) for transfer to the Oracle Financials general ledger system.

Oracle Financials

If the financial application is Oracle, a new online form can be accessed in the RMS, called GL Cross Reference. This form is the tool in which you can map the chart of account information (department, class, subclass, location, transaction code, line type and cost retail flag) to Oracle debit and credit information. By entering data in this form, you will load the following Oracle staging table:

FIF_ORA_CROSS_REF

The data in this Oracle staging table is used for accounting purposes within the Oracle Financial Application.

RMS Currency Interface from Oracle Financials

This process bridges new or updated currency information (code and rate information) from Oracle Financials to the RMS's currency tables.

RMS Validation Process

Currency information is staged to RMS staging table, FIF_CURRENCIES and FIF_CURRENCY_RATES. The program FIFCURU2107.PC then populates its two base tables, CURRENCIES and CURRENCY_RATES, based on what is received in this table. If the record cannot be processed due to validation failure or insert/update failure, a row will be written to the IF_ERRORS table, and the record will remain staged.

The nightly process FIFPDLP107.PC needs to run in order to pass the updated mapping fields to the RMS. This non-restartable program determines which financial application is bridged, based on the financial application flag maintained in the system options.

Oracle Financials

If the financial application is Oracle Financials, an Oracle to the RMS loading package ORA2RMS107B is called by the program FIFPDLP107.PC. This process loads the RMS staging table FIF_CURRENCIES. The data in this RMS staging table is used as a basis for populating the RMS base currency tables.

Scheduling

The process FIFPDLP107.PC should be scheduled in phase 1 of the RMS nightly process, prior to any other financial interface processes. FIFCURU2107.PC should run immediately after FIFPDLP107.PC.

The arguments for loading currency rate information on FIFPDLP is "in get_curr".

Restart

The programs FIFPDLP107.PC and FIFCURU2107.PC are single-commit, non-restartable processes. If an error is encountered upon execution, it performs a rollback.

RMS Error Table & Package in Interface to Oracle Financials

This package, IFSQLB.PLS, is called from various Platform Dependent Loading Processes throughout the entire Interface. This package contains the function INTERFACE_SQL.INSERT_INTERFACE_ERROR. This function inserts a record to the IF_ERRORS table.

RMS Error Process

The rows written to IF_ERRORS are descriptive. A record from IF_ERRORS will resemble the following:

- RMS2ORA 29-MAR-98 Currency code: USD
- Record locked on fif_currencies. Unable to update.

This row tells the maintainer of the IF_ERRORS table that an error occurred on the RMS to Oracle PDLP, and that the error happened on March 29, 1998. The record failed on the record containing USD. The reason the record failed was because the record was locked.

When interfacing processes runs encounter an error when inserting, updating, or for other reasons, a record will be written to the IF_ERRORS table. By using the package to handle these errors, users can allow a process to continue on to the next record, rather than failing the process entirely. In theory, an interfacing process will only fail if the INTERFACE_SQL.INSERT_INTERFACE_ERROR function fails.

Oracle Financials-Specific Loading Process

The RMS does not maintain this IF_ERRORS table in any way. Records written to the IF_ERRORS table will remain there until they are manually deleted. For this reason, it is recommended that a user be specified to maintain this table. Maintenance should be performed daily, and duties include:

- Correcting data that may have been staged incorrectly. This information must be corrected on the RMS staging tables so that it can be processed the next day through the PDLP process.
- Deleting records from IF_ERRORS.

Oracle Financials

If the financial application is Oracle, the PDLPs and validation processes involved will write to IF_ERRORS when errors are encountered during validation or transfer.

RMS GL Interface to Oracle Financials

This process bridges daily and monthly RMS stock information to Oracle Financials' general ledger system.

RMS Staging Process

There are three levels of Stock Ledger information that can be interfaced to the RMS GL Staging Table (and finally to the Oracle Financials API). They are:

- SKU-LOCATION Level (Daily)
- DEPT/CLASS/SUBCLASS-LOCATION Level (Daily)
- DEPT/CLASS/SUBCLASS-LOCATION Level (Monthly)

If either of the daily loads is required, a TRAN_DATA staging table must be created.

Run SALSTAGE.PC to load all new TRAN_DATA information to the staging table IF_TRAN_DATA.

The actual staging of this information is handled by one of the following programs:

- FIFGLDN1107.PC – Writes transactional (SKU-location) information to the RMS GL Staging Table FIF_GL_DATA. Only transaction codes entered into the CODE_DETAIL table under code type GLST are transferred.
- FIFGLDN2107.PC – Writes rolled (DEPT/CLASS/SUBCLASS-LOCATION) information to the RMS GL Staging Table FIF_GL_DATA. Only transaction codes entered into the CODE_DETAIL table under code type GLRT are transferred. The actual roll-up for the dept/class/subclass level is determined by the value of SYSTEM_OPTIONS.GL_ROLLUP.
- FIFGLDN3107.PC – Writes rolled (DEPT/CLASS/SUBCLASS-LOCATION) information to the RMS GL Staging Table FIF_GL_DATA. Only transaction codes entered into the CODE_DETAIL table under code type GLMT are transferred. The actual roll-up for the dept/class/subclass level is determined by the value of SYSTEM_OPTIONS.GL_ROLLUP.

Oracle Financials-Specific Loading Process

The nightly process FIFDLP107.PC must run in order to pass the GL data to Oracle Financials' APIs. This non-restartable program determines which financial application is bridged, based on the financial application flag maintained in the system options.

Oracle Financials

If the financial application is Oracle, an RMS to Oracle loading package RMS2ORA107B.PLS is called by the program FIFPDLP107.PC. This process loads the Oracle staging tables.

The data sent to the Oracle side staging table is created by mapping RMS's chart of account information to Oracle's accounting information created by the RMS form, GL Cross Reference (GUI). Information input on that form is then transferred to Oracle.

Scheduling

- FIFGLDN1107.PC should immediately follow SALSTAGE.PC.
- FIFGLDN2107.PC should immediately follow SALSTAGE.PC.
- FIFGLDN3107.PC is a monthly process that should be scheduled after SALMTH.PC because it requires the month end to occur in order to read the MONTH_DATA table.
- FIFPDLP107.PC should be scheduled to run after the given night's FIFGLDN1107.PC, FIFGLDN2107.PC, and FIFGLDN3107.PC (if it is a monthly close). There will be an Oracle process dependent on its completion in order to process the transactions in the Oracle Journal Generator.
- The arguments for loading GL information on FIFPDLP is "out get_gl_data".

Restart

The program FIFGLDN1107.PC is a fully-restartable process. If an error is encountered upon execution, it can simply be restarted.

The program FIFGLDN2107.PC is a fully-restartable process. If an error is encountered upon execution, it can simply be restarted.

The program FIFGLDN3107.PC is a fully-restartable process. If an error is encountered upon execution, it can simply be restarted.

The program FIFPDLP107.PC is a single-commit, non-restartable process. If an error is encountered upon execution, it performs a rollback.

RMS Interface to Oracle Financials – Initial Setup

There are several initial setup items for the conversion of the RMS-Oracle Financial Interfaces. They are specific to each financial package.

Notes: The information in this section is an overview only. Consult the Installation Guide for complete installation details.

Other tables will have to contain matching values in order for some bridging validation to work correctly. They are currencies, countries, states, and units of measure. For example, the unit of measure may be represented by "EA" in the RMS and "Each" in the financial interface. For interfacing purposes, these should be represented the same way.

If one application is running in production before the other is installed, other data must also be synchronized. These areas are payment terms, freight terms, and vendors. The interface processes can be used to transfer the data, but large volumes (particularly for vendors) may have to be bridged in parts.

Oracle Financials

1. Create a database link in the RMS environment over to the Oracle environment. The user within Oracle for this link must have insert/update/delete privileges.
2. Update the financial_ap indicator on system_options to O for Oracle, and the gl_rollup to the appropriate roll-up level (Dept, Class, or Subclass).
3. Create synonyms for the following Oracle objects to the RMS environment:

Tables:

- PO_HEADERS_INTERFACE_CAI
- PO_LINES_INTERFACE_CAI
- PO_SHIPMENTS_INTERFACE_CAI
- PO_DISTRIBUTIONS_INTERFACE_CAI
- AP_TERMS_EXPORT_CAI
- PO_FRT_TERMS_EXPORT_CAI
- PO_VENDORS_EXPORT_CAI

Synonyms to existing Oracle Tables:

- GL_CODE_COMBINATIONS
- GL_CODE_COMBINATIONS_CAI_V
- GL_SETS_OF_BOOKS
- GL_PERIODS
- GL_INTERFACE
- PO_VENDOR_SITES
- PO_VENDOR_CONTACTS
- AP_TERMS
- PO_LOOKUP_CODES

Views:

- AP_TERMS_V
- AP_VENDORS_V
- GL_DAILY_CONV_RATES_CAI_V-- It is a VIEW (not a synonym) in the apps_appdemo environment.

The following packages must be seen as well:

FND_CLIENT_INFO.SET_ORG_CONTEXT

Population of the fif_ora_gl_setup table

This table must be hard-inserted with some default values created within Oracle. (All tables referenced are Oracle owned and information regarding creation of referenced entities can be found in Oracle documentation.)

Set_of_books_id: Should contain the ID number of the accounting set of books the Retek transactions will write to.

Last_update_id: Should contain the employee ID of a valid employee from the hr_employees table. This employee should have sufficient privileges to create Purchase orders for the Oracle organization and may be a generic user created for use exclusively in Retek transactions.

Org_id: Should contain the ID of the organization that the Retek system corresponds to. This value is used to view data through Oracle's partitioned views.

sequence1_desc - sequence10_desc: These columns should contain descriptors for each segment of an account number. They will be used as labels in the General Ledger cross-reference form for both debit and credit information.

Category_id: Should contain the ID number of a valid category from the mtl_categories table. This value will be used as the default item category for all Purchase Order lines that are bridged to Oracle. Thus, it should be a generic "Retek" category.

Deliver_to_location_id: Should contain the location_id of a valid Oracle ship-to location on the hr_locations table. This value will be used as a default deliver-to location for all Purchase order distribution lines that are bridged to Oracle.

Deliver_to_organization_id: Should contain the corresponding inventory_organization_id from the deliver-to location above (from the hr_locations table). This value will be used as a default deliver-to organization for all distribution lines and the default ship-to organization for all shipment lines that are bridged to Oracle in the Purchase order feed.

RMS Purchasing Interface to Oracle Financials

This process bridges new or updated RMS purchasing information for transfer to Oracle Financials' purchasing system.

RMS Staging Process

The actual staging of this information is handled by a process that reads from a notification table, FIF_ORDHEAD. This notification table is written to during an existing batch process, ORDREV.PC. Changes are only bridged to FIF_ORDHEAD if the order specified has been modified and is Approved, Cancelled, or Completed.

Oracle Financials-Specific Loading Process

The nightly process FIFDLP107PC must run in order to pass the updated mapping fields to Oracle Financials. This non-restartable program determines which financial application is bridged, based on the financial application flag maintained in the system options.

Oracle Financials

If the financial application is Oracle, an RMS to Oracle loading package RMS2ORA is called by the program FIFPDLP107.PC. This platform specific process selects order information from RMS base tables. The order numbers selected are based on the order number staged on fif_ordhead. This process (which also handles the receiving data) will load the following Oracle staging tables:

- PO_HEADERS_INTERFACE_CAI
- PO_SHIPMENTS_INTERFACE_CAI
- PO_DISTRIBUTIONS_INTERFACE_CAI
- PO_LINES_INTERFACE_CAI

Scheduling

The process FIFPDLP107.PC should be scheduled in the RMS nightly process, after ORDREV.PC and any purchasing processes. It will pass receipt information required by other bridged processes, such as receipts.

The arguments for loading purchasing information on FIFPDLP is "out get_order".

Restart

The program FIFPDLP107.PC is a single-commit, non-restartable process. If an error is encountered upon execution, it performs a rollback.

RMS Receiving Interface to Oracle Financials

This process bridges new RMS receiving data for transfer to Oracle Financials' Accounts Payable system for use in invoice matching. This interface will only be used if the financial application accepts such a feed.

RMS Staging Process

The actual staging of this information is handled by PREPOST.PC, an existing batch program that, when called in a Post-fifgldn1 mode, will populate the receiving staging table FIF_RECEIVING. It will call this logic if the financial application is one that requires receipt information, based on the System Options Financial Application field.

Note that in this generic batch process, the interface-related tables fif_receiving and if_tran_data are both dropped nightly and then re-created with data. This is done for performance reasons because, for example, the if_tran_data table could contain millions of rows. If this creates issues with table sizing, this batch code can be customized to include special sizing parameters. Also note that the username/password combination must have the proper authority to drop and re-create these objects.

Oracle Financials

If the financial application is Oracle, an RMS to Oracle loading package RMS2ORA107B.PLS is called by the program FIFPDLP107.PC. This platform specific process selects receiving information from RMS base tables. The order numbers selected (which are the basis for selecting receiving data from RMS) are based on the shipment number staged on fif_receiving. This process (working with the ordering process) will load the following Oracle staging tables:

- PO_HEADERS_INTERFACE_CAI
- PO_SHIPMENTS_INTERFACE_CAI
- PO_DISTRIBUTIONS_INTERFACE_CAI
- PO_LINES_INTERFACE_CAI

If the receipt could not be bridged, an error record will be written to the interface error table IF_ERRORS.

Scheduling

See Scheduling for FIFPDLP.

RMS Terms Interface from Oracle Financials

The terms interface between Oracle Financials and RMS will provide a means to transfer payment terms and freight terms information originating and maintained in Oracle Financials. New terms information will be sent via a Platform Dependent Loading Process (PDLP) to RMS staging tables.

RMS Staging Process

The staging process begins with the PDLP, a batch program that reads information from the Oracle Financials's terms staging tables into the RMS staging tables FIF_TERMS and FIF_FREIGHT_TERMS. After the staging tables are populated, a validation program FIFTRMU2107.PC validates the fields and, if the data is valid, inserts the data into the RMS base tables. An error message will be written to the interfacing errors table IF_ERRORS for records that are not fully validated.

Oracle Financials

In the case of Oracle to the RMS, the loading package ORA2RMS107B.GET_TERMS is called by the program FIFPDLP107.PC.

The package call is based on the System Options Financial Application field. This package will load the FIF_TERMS and FIF_FREIGHT_TERMS tables.

Scheduling

The program FIFPDLP107.PC should be executed as part of the RMS nightly process, early in phase 1, before the Vendor Interface, because a new vendor may contain this new term as its default. FIFTRMU2107.PC should run immediately

The arguments for loading terms and freight terms information on FIFPDLP is "in get_terms".

Restart

The programs FIFPDLP107.PC and FIFTRMU2107.PC are single-commit, non-restartable processes. If an error is encountered upon execution, they perform a rollback.

RMS Vendor Interface from Oracle Financials

This process bridges new or updated vendor information to the RMS's supplier tables.

RMS Validation Process

Vendor information will be staged to two RMS staging tables, FIF_SUPS and FIF_SUPS_ADD. The RMS then populates its two base tables, SUPS and ADDR, based on the successful validation of this bridged data. If the data is incomplete, a record will be written to the interface errors table IF_ERRORS and the row will continue to be staged until it is overlaid by the PDL process (see below) or is removed by DBA intervention.

The process that performs this validation is VENDVAL107B.PLS. It is executed by the program FIFVNDU2107.PC.

Note: Vendor data may also be bridged and validated online via the RMS conversation. This is done from the main menu: Control – Supplier – Upload.

The nightly process FIFPDLP107.PC must be run in order to pass the updated mapping fields to the RMS. This non-restartable program determines which financial application is bridged, based on the financial application flag maintained in the system options.

Oracle Financials

If the financial application is Oracle Financials, an Oracle to RMS loading package ORA2RMS107B is called by the program FIFPDLP107.PC. This process loads the FIF_SUPS and FIF_SUPS_ADD RMS staging tables. The data in these RMS staging tables are used as the basis for populating the RMS base vendor tables.

Scheduling

The process FIFPDLP107.PC should be scheduled in phase 1 of the RMS nightly process, after the currency and terms loads. FIFVNDU2107.PC should immediately follow FIFPDLP107.PC.

The arguments for loading vendors information on FIFPDLP is "in get_vendors".

Restart

The programs FIFPDLP.PC and FIFVNDU2.PC are single-commit, non-restartable processes. If an error is encountered upon execution, they perform a rollback.

Oracle Financials-Specific Loading Process Batch Program

The process FIFPDLP107.PC is a financial application specific loading process. It performs the actual two-way information loading by calling financial-application-and- platform-specific packages, which loads the Oracle Financials's information to the RMS staging table or loading of RMS information to the Oracle Financials's staging table based on the two command line arguments. The first argument "in" or "out" means that the information is flow in to the RMS, or flow out of the RMS. The second argument is the function name.

As an example, if the financial application is Oracle Financials:

- to bridge currency information:
 >Fifpdlp107 username/passwd in get_curr
- to bridge PO information:
 >fifpdlp107 username/passwd out get_order

The actual scheduling of process fifpdlp107.pc should follow the instructions in each functional area.

Functional Area	Function Name
currency	Get_curr
term	Get_terms
Vendor	Get_vendor
Purchasing	Get_order
GL	Get_gl_data

RMS Invoice Matching to Oracle Financials

When Invoices are passed from RMS to Oracle Financials – Payables , the interface holds the line_type e.g Item,tax,freight,Misc.,etc. which is basically mapped to the Invoice Distribution Type like Item, Tax, Freight, Miscellaneous,etc.

The Line_type field of Fif_ora_cross_ref table is mapped to the Rms_line_type field of Fif_line_type_xref table.

When an Invoice holds ‘Tax’ as one of the line_type, then a tax code is required for the interface to work seamlessly. The Tax code is populated by the field vat_code in the AP_EXPENSE_REPORT_LINES_ALL table. One also needs to set-up a Tax Name in Oracle Payables. The path to set-up the Tax Name in Oracle Financials is as follows:

Oracle Payables -> Setup -> Tax -> Names

One needs to define the Tax Name, Tax Type, Rate, Description and the GL Account. The GL Account should be the same account which will correspond to the CCID mapped via the RMS GL Cross Reference Form.

The Batch program fifinvcu107.pc is associated for the invoice to be sent across to the interface tables and then the Oracle Financial Invoice Import processes are run to be available in Oracle Payables – Invoices.

HTS

Functional Overview

The HTS batch component is executed in order to insert or update data from the Customs Agencies.

The Customs Agencies will publish Harmonized Tariff Schedule information two or more times per year. This data is the basis for Item classification and duty calculation for Customs Entry. The batch component will insert new HTS codes and associated details, as well as update existing HTS data throughout the system.

Module Flow and Scheduling

The processing of Customs Agencies' information (htsupld) should be run two or more times per year depending on the number of updates published by the Customs Agencies.

The following diagram 1 outlines the individual modules that are associated with the HTS dialog:

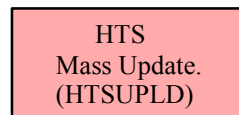


Diagram 1

Inventory Adjustments

Functional Overview

The Inventory Adjustment Upload (invaupld) module processes inventory adjustments initiated in an external system (i.e. WMS or POS) and makes the required adjustments in the RMS. There are two types of adjustments that can be made – adjustments to total stock on hand or adjustments to unavailable inventory. Adjustments to total stock on hand remove the inventory for the specified item/location from the system. Adjustments made to unavailable inventory do not remove stock on hand for the specified item, but do adjust the unavailable inventory for the item/location. Transaction-level stock ledger records are also written for each adjustment.

If the RMS system is interfacing with the Retek warehouse management system (ie. RDM), then the Inventory Adjustment Upload Conversion (lifinvup) module must run before the invaupld module. The conversion module will reformat the RDM inventory adjustment file into a format that is acceptable to the invaupld module.

The Inventory Adjustment Purge (invaprg) module will delete all obsolete inventory adjustment records where a pre-determined number of months have elapsed. The pre-determined number of months is a system option.

Module Flow and Scheduling

The Inventory Adjustment Upload (invaupld) module is designed to be executed daily. However if inventory adjustment information is required more frequently it can be scheduled to run multiple times throughout the day.

If interfacing with RDM, then the lifinvup module needs to run before the invaupld module in order to convert the RDM formatted file into a file acceptable by the RMS inventory adjustment module.

The Inventory Adjustments Purge (invaprg) module can be scheduled to run at any time; however, practical usage is to execute it only once at the end of each month.

The following diagram 1 outlines the individual programs that are associated with the Inventory Adjustments dialog:

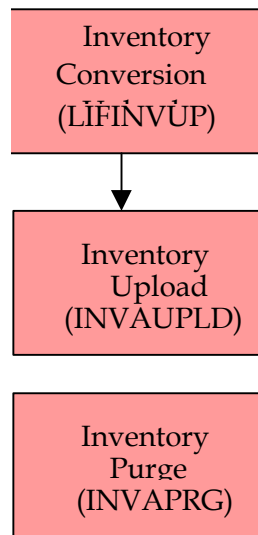


Diagram 1

Invoicing

Functional Overview

The Invoice Matching component contains several batch modules.

Invclshp

This batch program will close all shipments that have remained open for a specified number of days and are not associated with any open invoices.

Invcpst

This batch program will insert records into the IIF (invoice interface to financials) staging tables.

Invmatch

This batch program will perform the automatic invoice matching process.

Ediupinv

This batch program will accept EDI invoices and credit notes and load them into the system after validating all information.

Edidldeb

This batch program will download EDI debit memos and credit note requests into a flat file for interface to the vendor.

Invprg

This module will purge posted invoices after a specified number of history months. It will be responsible for non-merchandise invoices and stand-alone credits/debits as merchandise invoices will be purged with their associated POs.

Module Flow and Scheduling

Current version of Invoice Matching includes the following batch modules:
Invoice Close Ship (invclshp), Invoice Match (invmatch), Invoice Post (invcpst), EDI upload (ediupinv), EDI debit memo/credit note request download (edidldeb) and Invoice Purge (invprg).

The following diagram 1 outlines the individual modules that are associated with the Invoice Matching dialog:

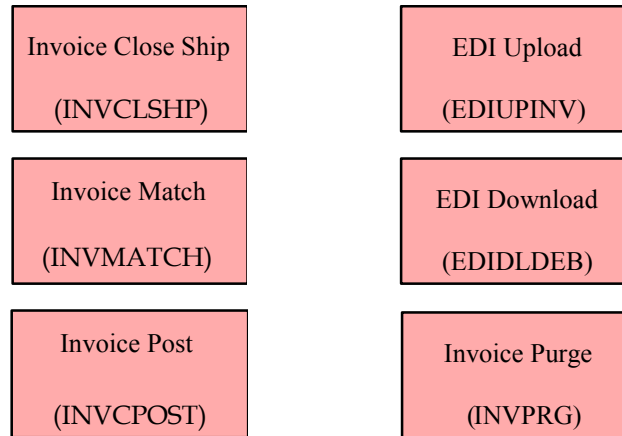


Diagram 1

Item Maintenance

Functional Overview

The POS Download (posdnld) module is executed to write temporary and permanent price modifications to a file to be used by a point of sale system.

The Item Download (itemdnld) module will download new, modified, or deleted items maintained in RMS to the warehouse management system.

Module Flow and Scheduling

The POS download module should be run daily, after the pricing, promotion, and clearance modules: price change download (pcdnld), price change clearance reset download (pccrdnld), price change clearance download (pccdnl), and promotion extract (prmext).

The Item Download module should be run daily to provide the warehouse with the most up to date item information. This program can be run at any point in the batch schedule, but it is recommended to be run during the first phase.

The following diagram 1 outlines the individual modules that are associated with the Item Maintenance dialog:

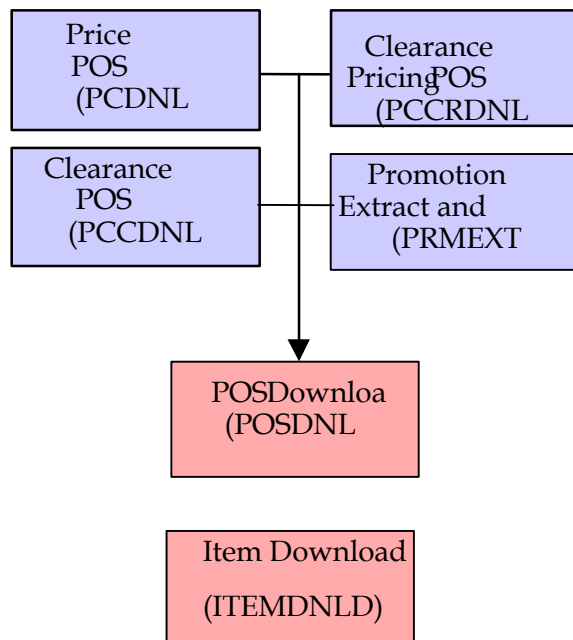


Diagram 1

Item Reclassification

Functional Overview

The Item Reclassification batch module is executed in order to reclassify items from one department, class or subclass to another. This reclassification of items into different merchandise hierarchy level is initiated or requested online in the item reclassification dialog, with an effective date specified. The actual reclassification of the item is processed in the batch for all reclassifications taking effect the following day.

When an item is reclassified, stock ledger transactions are written to move inventory amount associated with this item from the old merchandise hierarchy level to the new one in the stock ledger. If there are active orders for this item, OTB will also be updated. Pos_mods records are written for downloading to stores. History, such as sales history, will NOT be moved.

Module Flow and Scheduling

The Item Reclassification (reclsdly) module is scheduled to run daily during Phase 4 of the batch schedule. The requirement is for all the batch modules that update inventory and retail / cost prices to be completed before reclassification takes place.

The Item Reclassification (reclsdly) module is the only module associated with the Item Reclassification dialog (see the following diagram):

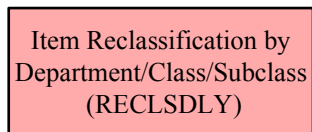


Diagram 1

Layaway

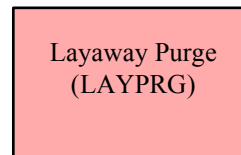
Functional Overview

The only batch component in the Layaway dialog is the Layaway Purge module. The function of this module is to delete old layaway accounts that have exceeded the retention period. The retention period is maintained online in the System Options dialog, by the system administrator. Layaway accounts can only be deleted if paid in full, refunded, or returned to stock, and the sale is older than twice the number of days allowed for completion of a layaway contract.

Module Flow and Scheduling

The Layaway Purge (layprg) module can be scheduled to run at any time; however, practical usage is to run it only once at the end of each month near the end of the batch cycle.

The following diagram 1 outlines the individual program that is associated with the Layaway dialog:



Dialog 1

Letter of Credit

Functional Overview

The Letter of Credit (LC) batch components are executed in order for a user of the RMS to communicate letter of credit information to and from banks.

The Letter of Credit Application Download (lcadnld) module is run to download an LC to the appropriate bank.

The Letter of Credit Confirmation Upload (lcupld) module is run to upload confirmation from the bank that the LC has been approved.

The Letter of Credit Amendment Download (lcmdnld) module is run to download amendments made to an LC to the appropriate bank.

The Letter of Credit 798 Upload (lcup798) module is run to upload drawdowns and charges made against the LC by the bank.

Module Flow and Scheduling

The LC batch components should be scheduled to run at any time. The order in which the programs should be run is LC Application Download (lcadnld), LC Confirmation Upload (lcupld), LC Amendment Download (lcmdnld), and then LC 798 Upload (lcup798). Most likely each will be run in the nightly batch cycle.

The following diagram 1 outlines the individual programs that are associated with the Letter of Credit dialog:

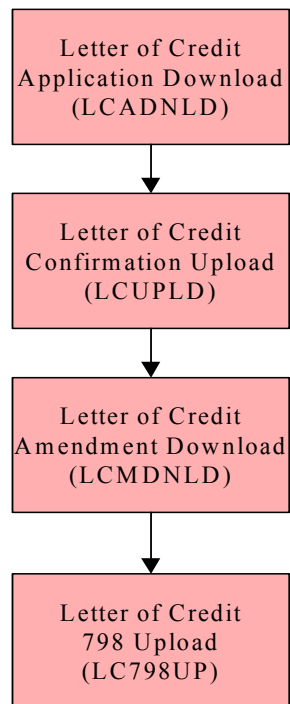


Diagram 1

Masks

Functional Overview

The Mask batch modules provide an interface to an outside planning system by accepting planned sales data at a merchandise hierarchy/store level into the Retek database. These figures are stored on the department, class or subclass sales history tables and are used when creating plan based masks online.

The Class Level Planned Sales Units Upload (plncupld) module is run to update the class sales history table with planned sales units imported from an outside planning system.

The Department Level Planned Sales Units Upload (plndupld) module is run to update the department sales history table with planned sales units imported from an outside planning system.

The Subclass Level Planned Sales Units Upload (plnsupld) module is run to update the subclass sales history table with planned sales units imported from an outside planning system.

Module Flow and Scheduling

The Mask modules can be scheduled to run ad hoc any time new planned sales figures become available. The modules are independent of each other and all other batch processing.

The following diagram 1 outlines the programs that are associated with the Mask dialog:

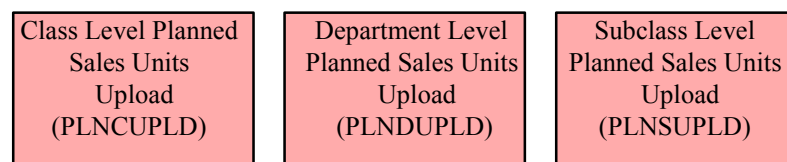


Diagram 1

Open to Buy Maintenance

Functional Overview

The Open to Buy (OTB) batch modules provide various interfaces to and from an external OTB planning system. Their main functions are to download data from the OTB and stock ledger tables for use by the external OTB planning system, and to accept budget and forward limit figures from that system into the Retek database.

The Open to Buy Upload (otbupld) module accepts new budget data or budget adjustments from an external OTB planning system. This data is used to update the current budget figures or create new budget records.

The Open to Buy Download (otbdnld) module extracts OTB data (excluding budget figures) and write it to a file to be used by an external OTB planning system.

The Forward Limit Percentages Upload (otbupfwd) module uploads forward limit percentages by period from an external OTB planning system.

The On-Order Extract (onordext) and Stock Ledger Data Download (stlgdnld) modules export stock ledger data, such as actual inventory and sales, etc. as well as on-order amounts to a file on a weekly or historic basis. This information is used by the external OTB planning system.

The Outstanding Order Download (otbdlord) module sums outstanding order amounts from past periods for each subclass and exports the data to a file for use by an external OTB planning system. This program is different from the On-Order Extract module in that it will only extract on-order information for a subclass. The On-Order Extract module extracts on-order information for all subclass/location combinations.

The Open to Buy Purge (otbpgrg) module is run to delete information that is at least one half year old. The current and previous half year's purchase budget data is retained.

Module Flow and Scheduling

It is suggested that the download modules be run in the standard download portion of the batch schedule (i.e. Phase 4) and the upload modules in the standard upload portion of the batch schedule (i.e. Phase 2), however, they can be run at other times during the batch schedule. The two modules with an actual dependency are the On-Order Extract and the Stock Ledger Extract modules. Since the Stock Ledger Extract module downloads on-order information, the On-Order Extract program should run before the Stock Ledger Extract module.

Since OTB numbers are by week, it is generally recommended that the modules be run on a weekly basis, however, if needed, they can be scheduled to run throughout the week as well.

The OTB Purge module should be run as the last module since it is removing information that may still need to be downloaded. Also, since it purges halves, it should probably be run at the end of month instead of end of week.

The following diagram 1 outlines the individual modules that are associated with the Open To Buy Maintenance dialog:

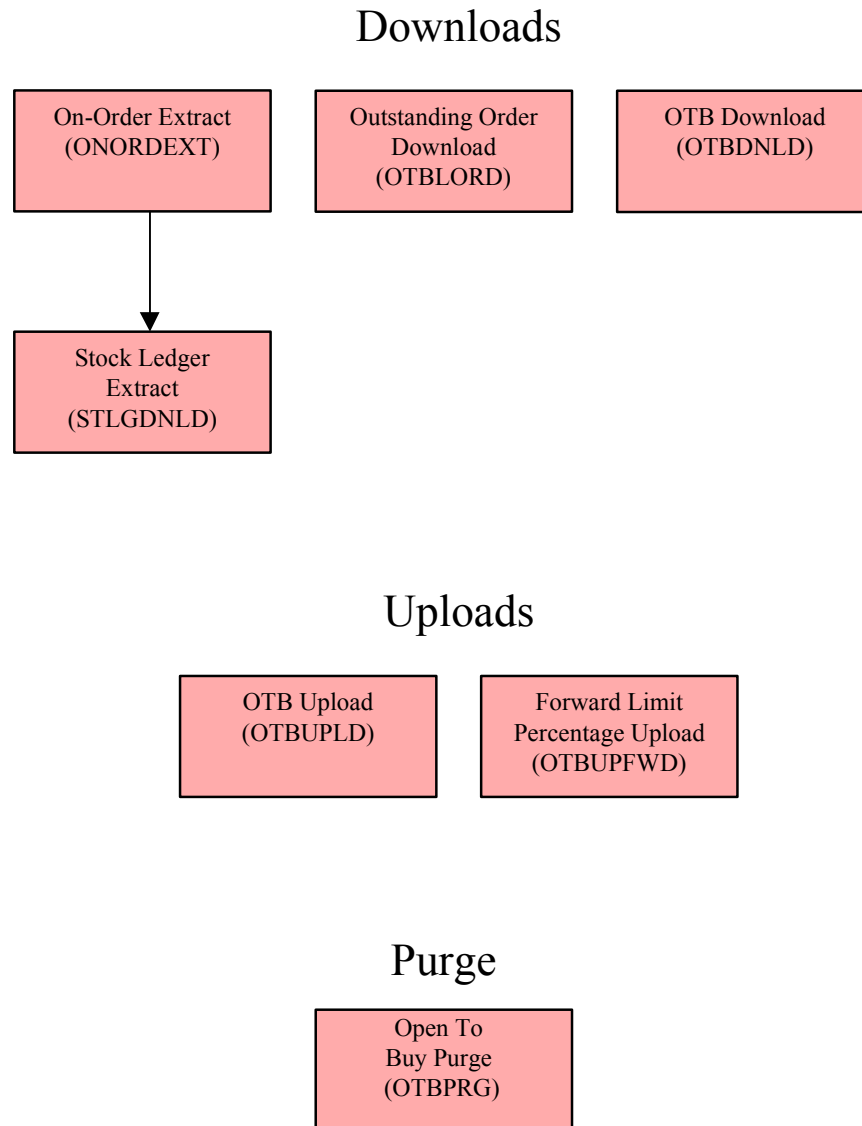


Diagram 1

Organization Hierarchy

Functional Overview

The Store Ship Schedule Purge module deletes old store ship schedule records and warehouse blackout records that have exceeded the retention period. The retention period is maintained online in the System Options dialog.

The Location Download (locdnld) module will download store and warehouse information to a warehouse management system.

The Store Add (storeadd) module will create new stores in the RMS system. When a new store is created in the on line dialog, the store is saved to a temporary table. This module will process the newly added stores found on the staging table and create a new store in RMS.

Module Flow and Scheduling

Store Shipment Schedule Purge (schedprg) can be scheduled to run at any time; however, practical usage is to run it only once at the end of each month.

The Location Download (locdnld) module can run at any point in the batch schedule, however it is recommended to run every day in Phase 1.

The Store Add (storeadd) module can run at any point in the batch schedule, however it is recommended to run every day in Phase 1.

The following diagram 1 outlines the program associated with the organization hierarchy dialog:

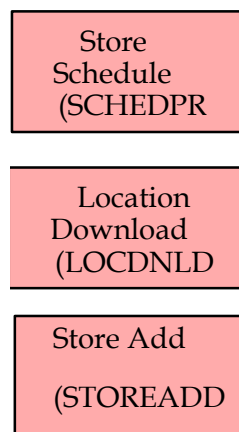


Diagram 1

Pricing (Retail and Cost)

Functional Overview

The primary functions of the Pricing batch component are to download price changes to the point of sale (POS) system and to update the RMS tables with new prices. Note: Only retail price changes are downloaded to the POS system. Multiple programs are necessary to perform these functions. These programs are described below.

The Price Change POS Download (pcdnld) module selects permanent price change records, which are set to go into effect a pre-determined number of days after today. The details of these records are then written out to a RMS table, which is used as an interface point to with the point of sale system. The number of days before the price change goes into effect that the details are downloaded to the POS system is determined by a system option. After this program has processed a price change, the status of the price change is changed to Extracted.

The Retek Price Change Extract (pcext) module selects permanent price change records, which are set to go into effect tomorrow and updates the RMS item/location and item/zone tables with the new prices. In addition to the updates to the retail prices in the system, records are also written for price history and transaction-level stock ledger. Once this program has processed a price change, the status of the price change is changed to Delete.

The Price Change for Store Zone Changes (pctran) module processes pricing transactions that are created when a store is moved from one zone to another. This program will select all items that exist at the store that was moved and are associated with the zone group in which the move was made. Then, updates are made to the item/store prices, along with insert into price history, supplier history, and transaction-level stock ledger tables. Also, records are written out to the POS interface table, similar to permanent price changes, such that the store(s) affected will be notified.

There are three other batch programs that deal with permanent retail price changes. They are Price Change Impact (pcimpc), Price Change Overlap (pcovrl), and Purge Price Changes (pcprg). Price Change Impact updates the impact quantity and impact amount fields on the price change header table to take into account the impact of the price change, in both dollars and units, based on the current stock on hand. The Price Change Overlap program searches through all submitted price change records and checks for conflicts with other price changes in submitted, approved or extracted status. Conflicts that are found in this program are written to a price change conflicts table that can be used for reporting purposes. The Price Change Purge program is responsible for removing old price changes from the system. Price changes are removed from the system using the following criteria:

- the status of the price changes is Canceled or Deleted
- the status is Extracted, but the effective date was yesterday
- the status of the price change is Rejected and the effective date of the price change has met the requirement for the number of days that rejected price changes are held.

Note: The number of days that rejected price changes are held is determined by a system option.

The Retek Supplier Cost Changes Update (sccext) module selects supplier cost change records, which are set to go into effect tomorrow and updates the RMS item/supplier table with the new cost. The item/location tables are also updated with the new cost if the cost change is a base cost change (supplier is the primary supplier for the item). In addition to the updates to the unit costs on the item/location tables when the base cost changes, records are also written for price history, supplier history and transaction-level stock ledger (if using standard cost). The costs on approved orders may also be updated if the recalculate order indicator is set to Yes for the item/supplier combination. The Pre/Post program for multi-threadable programs should be run after this program to update the status of the cost change to Extracted.

The final pricing batch program is the Cost Event Purge (ccprg) module. This program is responsible for removing old cost changes from the system. Cost changes are removed from the system using the following criteria:

- the status of the cost change is Delete, Canceled, or Extracted
- the status of the price change is Rejected and the effective date of the cost change has met the requirement for the number of days that rejected cost changes are held.

Note: The number of days that rejected price changes are held is determined by a system option.

Module Flow and Scheduling

The Price Change POS Download (pcdnld) module is run on a daily basis during Phase 1 of the batch schedule.

The Price Change for Store Zone Changes (pctran) module and the Retek Price Change Extract (pcext) module are also run daily during Phase 3 of the batch schedule.

Price Change Impact (pcimpc), Price Change Overlap (pcovrl), Purge Price Changes (pcprg), Cost Event Purge (ccprg), and Retek Supplier Cost Changes Update (sccext) modules should be run as needed; however, practical usage is to run them once a day after all the batch cycles have completed.

The following diagram 1 outlines the individual programs that are associated with the Pricing (Retail and Cost) dialog:

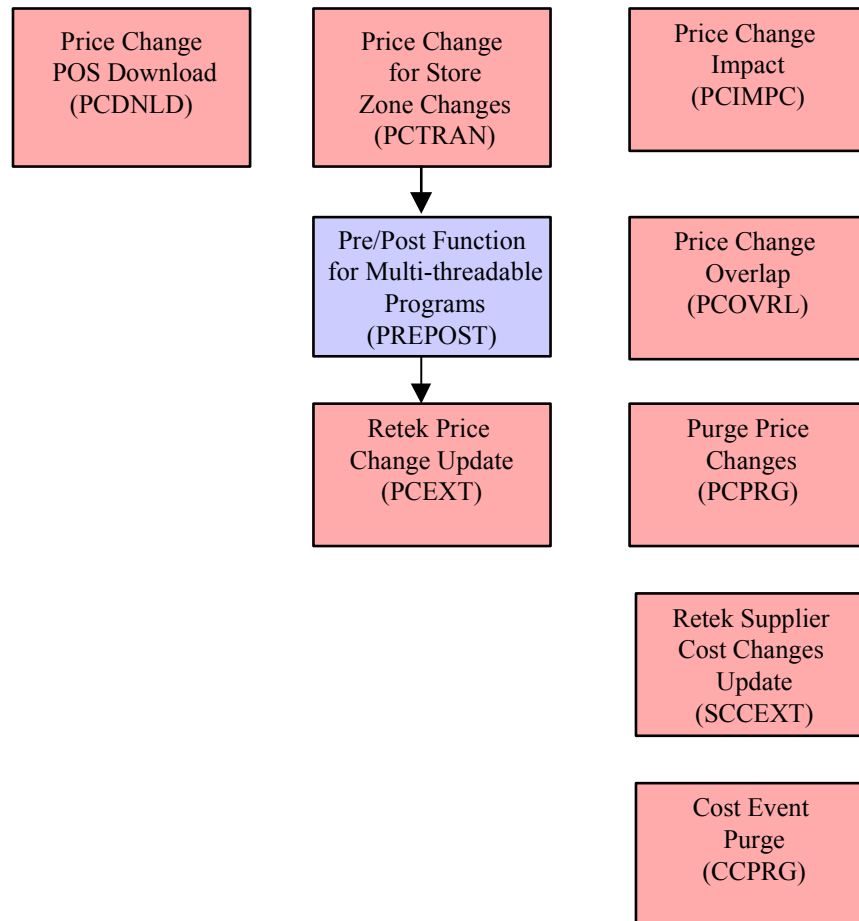


Diagram 1

Promotions

Functional Overview

The Promotions batch component is executed in order to create point of sale (POS) transactions that indicate an item's promotional price prior to the start of a promotion event and reset the item's price once the promotional event is complete. To create point of sale records, a promotion must have a status of 'Approved' and the start date of the promotion must equal the current date minus POS extract days (POS extract days is maintained in system options.).

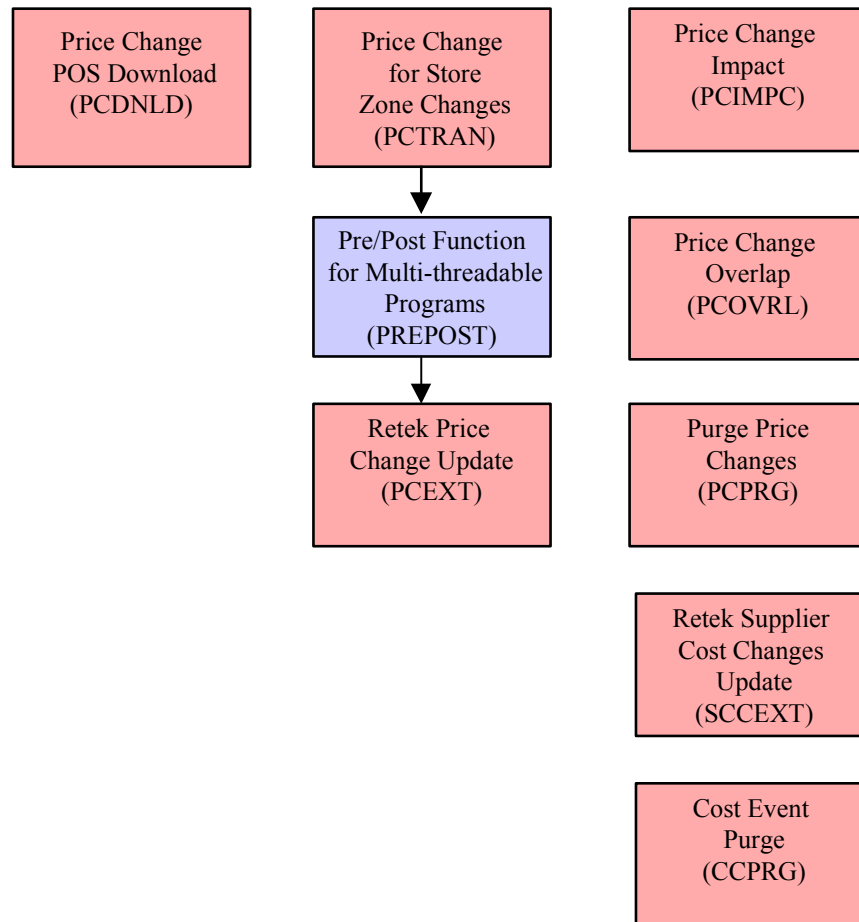
The Item Level Promotions Explode (prmxpld) module runs first to create item level records for promotions that have been setup at a department level. The Promotion Price Extract and Download (prmxext) module runs next to build item/location relationships and create the records to be sent to the point of sale. Next, if the multi-promotional indicator is set to Yes online, the Promotion Download (promdnld) module writes mix-n-match and threshold promotional data to a file.

The Pre/Post Functionality for Multi-threadable programs (prepost) module runs last to update the status of items that have been added or deleted to prevent sending duplicate records down to the point of sale. After an item has been extracted or deleted, the prepost module removes its AI (add item) or DI (delete item) status. This ensures that a new record will not be created for the same item the next time the Item Level Promotions Explode (prmxpld) module is executed. However, if an item is changed or reinstated after it has been extracted, the AI status will be returned and a new record will be created.

When the Promotion Price Extract and Download (prmxext) module is executed the day prior to the end of a promotion event, it will create a record that will set the item's price back to the current retail.

The Promotions Purge (prmprg) module will remove promotions from the system that have a status of Delete, Cancel, or Complete, and the number of months since the ending date of the promotion is greater than promotion retention period. The promotion retention period is maintained online in the System Administration dialog.

Module Flow & Scheduling



The Item Level Promotions Explode (prmxpld), Promotion Price Extract and Download (prnext) and Pre/Post Functionality for Multi-threadable programs (prepost) modules should be scheduled to run on a daily basis after the pricing functionality has completed. The order of execution is explained above.

The Promotions Purge (prmprg) module can be scheduled to run at any time; however, practical usage is to run it only once at the end of each month at the end of all batch cycles.

The following diagram 1 outlines the individual programs that are associated with the Promotions dialog:

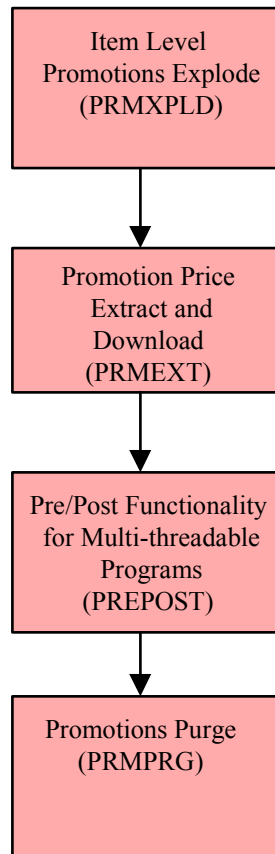


Diagram 1

Purchase Order

Functional Overview

The Purchase Order batch component is executed in order to maintain the ordering tables when orders are updated due to retail price changes, cost changes or deals application, and to keep track of changes made to EDI orders that will need to be sent to the vendor. Outdated orders are also removed from the system. The purchase ordering batch programs also provide a method of interfacing with a warehouse management system.

The Purchase Order Information Written to Order History Tables (ordrev) program will write versions of approved order to order revision history tables. When orders are approved or when approved orders are modified, this program selects order numbers from the rev_orders table and writes current order information to the order/allocation revision tables. After the new version has been written to the order revision tables, all records will be deleted from the rev_orders table for that order_no. This program processes order changes made by the client that may need to be sent to the vendor. The EDI Purchase Download (ediordnld) program extracts the revised orders to send to the order's supplier (additional information concerning this program can be found in the EDI section). The order changes should always be referred to as 'versions' and kept clearly distinct from order 'revisions' which are vendor changes uploaded via the EDI Vendor Acknowledgement Upload (ediupack) program (additional information concerning this program can be found in the EDI section).

The Purchase Order Information Written to Order History Tables program also allows the Retek Warehouse Management system and Retek to interface, by sending the warehouse PO and allocation (i.e. pre distribution) information to prepare the warehouse for incoming orders. The program will create two flat files, PO header and PO detail, based on approved orders found on the rev_orders table. The program will also create Pre Distribution Header and Pre Distribution Detail flat files, which will enable the warehouse to perform cross docking activities. The last file produced by the batch program is a component ticketing file that holds retail and ticketing information for non sellable pack items. This file allows the warehouse to correctly ticket the components of the pack item, before distributing the items to the stores. If an order is not in approved status at the time the batch program runs, then none of the above processing will occur. The record will stay on the rev_orders table until the PO is approved or deleted.

The Inbound Work Order (powodld) program is used to download work order details to the warehouse for processing incoming purchase orders and associated distributions (i.e. allocations).

The Retail Price Change on Purchase Orders (ordupd) module updates the retail costs on orders when a retail price change is made to an item that is on order. All items on order except for those on an approved order will be changed. Items on orders in Approved status will be updated only for the amount of items not yet received. Open to Buy, if set at the retail level, is also updated at this time to reflect the retail price change. Additional modules that update the order's cost/retail information include the Supplier cost Change (sccect) and the Order Discount (orddsct) programs. The Supplier Cost Change (sccect) module can update approved purchase orders if the update purchase order indicator is set to Yes for the supplier cost change. If it is, the cost on the order will be updated to reflect the new supplier's cost. A more detailed description of this module can be found in the Pricing Retail and Cost section. The Order Discount (orddsct) module is used to update order costs based on deals. A more detailed description of this module can be found in the Complex Deals Management section.

The Purchase Order Purge (ordprg) module removes old orders from the system. All details associated with an order are deleted when the order has been closed for more months than specified in the system's options. This program also creates a file of deleted orders to be downloaded to the warehouse system for deletion.

Module Flow and Scheduling

The Purchase Order Information Written to Order History Tables (ordrev) module can be run at any time, dependent on need. When run, it should be executed after the replenishment orders are built and before the EDI orders are transmitted to the supplier. More specifically, order revision should be run in Phase 4 before the EDI Download Orders module and after the EDI Location Address module.

The Inbound work order download (powodld) module should be run after the Purchase Order Information Written to Order History Tables program. This will allow the purchase order and associated work order information to be downloaded to the warehouse at the same time.

The Retail Price Change on Purchase Orders (ordupd) module should be run daily during Phase 4 after the Price Change Extract and Price Change Clearance Extract modules are run to ensure that the most recent price changes are applied to the orders.

The Purchase Order Purge (ordprg) module is a maintenance module that can be run at any time, but is generally suggested to be run monthly.

The following diagram 1 outlines the individual modules that are associated with the Purchase Order dialog:

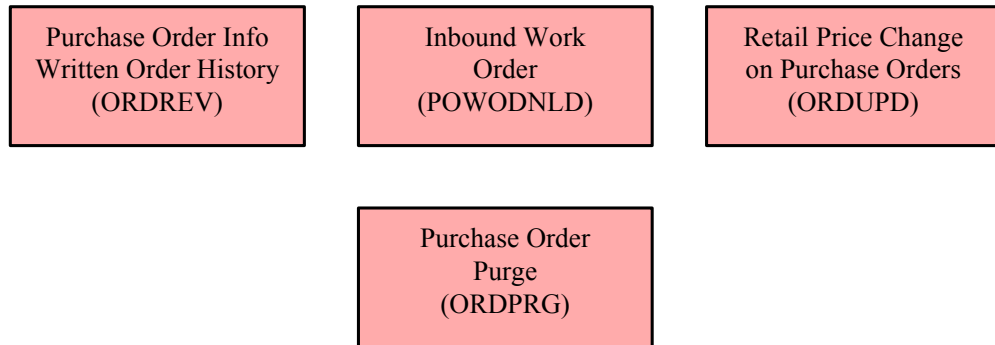


Diagram 1

Receiving (P.O)

Functional Overview

The Receiving batch processing consists of four modules : Advanced Shipment Notification Download (asndnld), Automatic Shipment Extraction (rcvext), P.O. Receipt Transactions Upload (rcvupld) and the Carton Receiving Upload for P.O. & Transfers (ctniupld).

The first of these creates an advanced shipment notification (ASN) file, if shipment information is required by the warehouse management system (or any other system that will process receipts from a supplier). This module is executed daily, creating shipment notifications based on RMS shipment/order information.

If the system that captures receipt information (e.g., WMS or POS) requires shipment information in order to process receipts, RMS will create shipment for orders from suppliers that do not send ASN information. To setup this automatic creation of shipment information, the user must check Automatic Shipment Indicator on the System Control form, rcvext module will then create shipment information for all approved orders that meet the following criteria:

- The supplier is not an ASN supplier (Note that this is maintained in the supplier maintenance dialog).
- The first day order delivery that will be accepted is within a given time period of today's date.
- Shipments have not already been created for the order.
- If interfacing with Retek's warehouse management system (ie. RDM), the rcvext module will not create warehouse shipments.

For each order, shipments and shipment-detail records will be created which can then be downloaded to the proper system (e.g., WMS). If the system that captures receipt information (e.g., WMS or POS) performs "blind receiving" (i.e., does not require shipment information to match with receipts) or if all receiving is handled within RMS, there is no need to execute this program.

The receipt transaction file created from an external source like WMS or POS, can contain receipts for purchase orders and transfers, as well as receipts processed with carton information. Therefore, prior to running rcvupld module, Transfer Receipts Upload (tsfiupld) module, and ctniupld module, a P.O. Receipt, Transfer Receipt, and Carton Receipt Parse (tsfparse) module must be run to split out the PO receipts, the transfer receipts and the carton receipts into three files to be processed by rcvupld, tsfiupld and ctniupld, respectively.

If interfacing Retek Distribution Management (RDM), then the Receiving Conversion (lifrcvup) module must run before the tsfpars module. The conversion module will reformat the RDM receiving upload file into a format that is acceptable to the tsfpars module.

The rcvupld module processes receiving detail information of a PO receipt captured by an external source. The receipt transactions provide feedback on existing orders, and cross-reference existing shipments. The receipt detail information can be processed using any of the supported item types (i.e. SKU, UPC, or VPN) within the RMS system.

The ctniupld module processes receipt transactions for carton shipments, which are created for either a PO or a transfer. If the carton shipment is for a PO, this module performs PO receiving logic for all items in the shipment, as in rcvupld module, and update the quantity received to be equal to quantity expected for all shipment detail records. Additionally, if the PO is a cross-dock pre-marked PO, this module also creates transfers along with their shipment records to transfer the goods from warehouse to the PO's final destination. In a DC flow through type of operation, Transfer Shipments Upload (tsfoupld) module should follow ctniupld module in order that the transfers created in ctniupld module be shipped out within the same day. Note that to process transfer out shipments in the tsfoupld module, an input file from the external system (i.e. WMS) is expected which contains the carton number.

The following functions will be performed for each received item :

- Create/update shipment and shipment/SKU records
- Create/update order/SKU and order/SKU/location records
- Update order header status to complete if all units received against the order
- Create item/location relation for receiving location (if it doesn't exist)
- Update perpetual inventory
- Update average cost of item at receiving location (if stock on hand not = 0 or negative)
- Write stock ledger financial transactions
- Update the snapshot stock on hand quantity for received item/location if a stock count is in progress
- Create transfers based on allocations tied to the order
- Update OTB table's receipt amount
- Write supplier data record for vendor analytics
- Update contract_detail and contract_header if PO is for a contract

- Write ticket_request if item is received or allocated into store and if the ticket is generated at PO receipt time
- Write AP_tran if the supplier is using “Evaluated receipt settlement” as payment process method and a matched shipment is found for the receipt

Module Flow and Scheduling

The asndnld module must run after the ASN Upload (ediupasn) module.

The rcvext module must run before the external system files for receiving are uploaded. It will include the orders created during the previous day’s batch cycle and any orders that were created on-line during the current business day.

If interfacing with RDM, the lifrcvup module must run before the tsfpars module in order to create the appropriate file format.

The rcvupld module will be run after the tsfpars module. It will likely be run at the beginning of the batch run during the POS polling cycle, or possibly at the end of the batch run if pending warehouse transactions. It can be scheduled to run multiple times throughout the day, as WMS or POS data becomes available.

The ctniupld module will be run after the Receipt Upload module runs. It can also be scheduled to run multiple times throughout the day, as WMS or POS data becomes available. In a DC flow through type of operation, tsfoupld module should follow ctniupld module in order that the transfers created in ctniupld module be shipped out within the same day.

The following diagram 1 outlines the individual modules that are associated with the Receiving dialog:

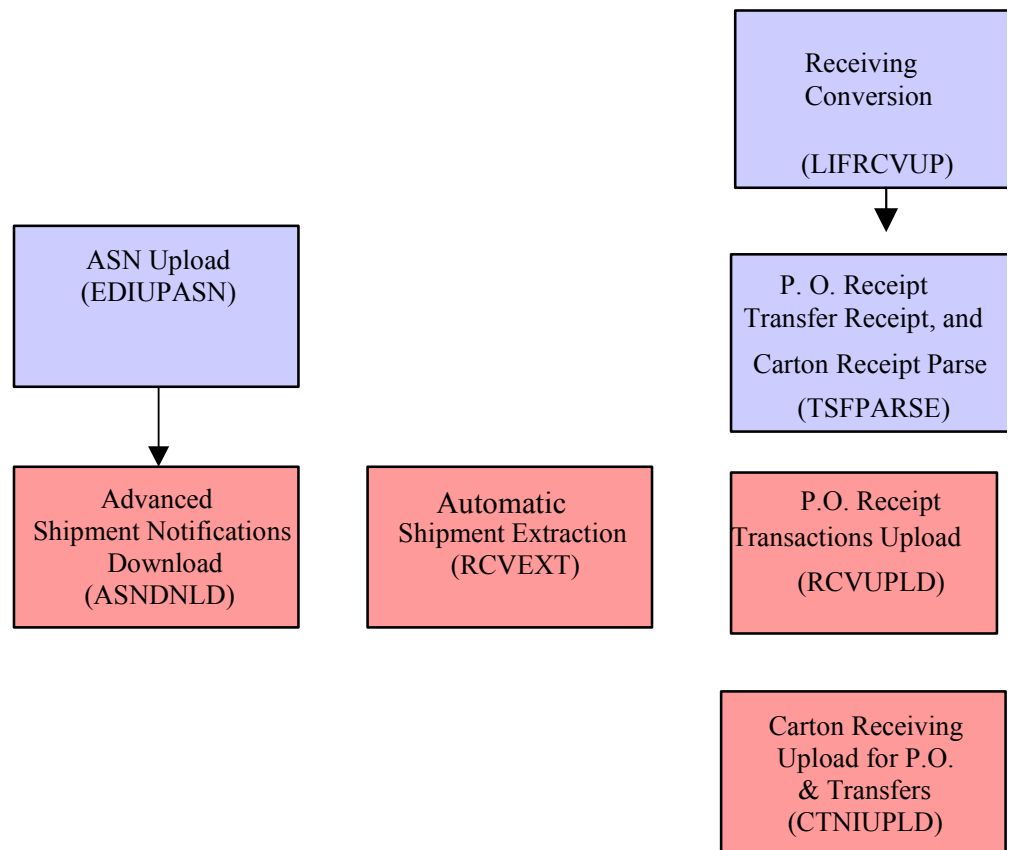


Diagram 1

Replenishment

Functional Overview

The Replenishment batch components are designed to manage stock levels.

Rplatupd

The batch program, Rplatupd, executes the logic necessary to maintain replenishment attributes for an item list. When the Replenishment Attribute Maintenance form is used to maintain the replenishment attributes for an item list, the form calls a package to explode the items in the item list and locations to two tables, REPL_ATTR_UPDATE_ITEM and REPL_ATTR_UPDATE_LOC. This batch program reads these tables and processes the item location relationships, using the information written to the table to determine what replenishment attributes for what locations have to be updated. SKU/locations can be activated on replenishment, deactivated from replenishment and have its replenishment parameters updated. The replenishment parameters are maintained for all SKU/locations on the REPL_ITEM_LOC and REPL_DAY tables. Items are processed in order of sequence number. Each item list maintenance submittal is given a sequence number.

Rpladjs/f

The Replenishment Maximum Stock Adjustment batch programs (rpladjs and rpladjf), recalculate the maximum stock levels for all fashion and staple SKU locations that are using a replenishment method of 'F' (floating point). The maximum stock level is calculated using the sales history (for stores) or the issues history (for warehouses) of various periods of time in order to accommodate seasonality as well as trend. For fashion SKUs, the sales history is obtained from the Fashion SKU Store Sales History (rag_skus_st_hist) table and the issues history is obtained from the Fashion SKU Warehouse Issues History (rag_skus_wh_hist) table. For staple SKUs, the sales history is obtained from the Staple Store Sales History (win_store_hist) table and the issues history is obtained from the Staple Warehouse Issues History (win_wh_hist) table. A summary of the calculations for SKU/stores follows:

Maximum Stock = Sales Rate + service level factor x SQRT(Sales Rate)

1 Year Ago

Today

1/1/96

1/1/97

|----- Z ----|----- X -----|----- Y ----|

64 60 56 52 48 44 40 36 32 28 24 20 16 12 8 4 0 4

8 12 ...

Service Level Factor = 1 if the user sets the service level to an 85% in-stock service level, or 2 if the user sets the service level to an 98% in-stock service level.

Week	1	2	3	4	5	6	7	8	9	10
Sales Regular Promotion										
	10	15	3			5	10	0	0	10
			20	25	20	10				

In both programs substitute SKUs are considered as well. If the SKU has substitute SKUs and their sales histories are being used to calculate the maximum stock level of the main SKU, the sales histories for the main SKU and substitute SKUs are summed across the time periods and their totals are used to determine the sales rate ratio.

If the calculated maximum stock level is less than the associated minimum stock level for the SKU/location being processed, the maximum stock level is set to the minimum stock level (i.e. the minimum stock level acts as a "safety net").

When calculating the maximum stock level for SKU/warehouse locations, warehouse issues are used. The calculations are similar to when calculating for stores except that all weeks in the time periods are taken into account since warehouse issues are not distinguished between regular, promotional and clearance.

Rplext

The Replenishment Extract program (rplext), is the driving program for the replenishment process. It cycles through every item-location combination that is set to be reviewed on the current day, and calculates the quantity of the item that needs to be ordered to the location (if any). The module reviews the item/location as defined within the item/location review cycle (Every day, multiple times per week, once every 2 weeks, once every 3 weeks, etc.). The supplier and origin country at the item/location level specify from where the item is sourced to meet the calculated demands of the location.

The Replenishment Extract (rplext) module creates either temporary purchase order records on the Order Temporary table (ORD_TEMP) and/or outputs replenishment information records on the Replenishment Results table (REPL_RESULTS), depending on how the order control parameter is set at the item/location level:

- If the order control is set to “Automatic”, temporary purchase order records are created with an approved status (when orders are actually built from these records, they are built in approved status).
- If the order control is set to “Semi-automatic”, temporary purchase order records are created with a worksheet status (when orders are actually built from these records, they are built in worksheet status).
- If the order control is set to “Manual”, replenishment results records are written to the Replenishment Results table against which reports can be run. Using the results from this table, the buyer may then manually create a purchase order (or perform another transaction) to meet the demand of the location.

The module distributes this Recommended Order Quantity (ROQ) over any applicable simple packs (sellable and orderable packs that contain only a single item – for instance, a six-pack of cola or a twelve-pack of socks).

The logic of this program is determined mainly by the stock category of the item-location combination. If the stock category is Direct to Store, the process is fairly straightforward. The ROQ is calculated, and this quantity is then distributed among any applicable simple packs.

If it is Crossdocked, the ROQ and distribution calculations are handled similarly to the Direct to Store category. However, when crossdocking the item must be ordered to a warehouse before being allocated to the store. This means that the total number of items being ordered must be tracked across all stores sharing the same source warehouse, so that a summary record can be written to ORD_TEMP to connect all the stores and create one order to the warehouse to be allocated from there.

No item-store combinations with a stock category of Warehouse Stocked are processed by this program. Since these stores are supplied via transfers from the source warehouse rather than purchase orders, they are processed in a separate program (reqext). However, orders must still be built so that the source warehouse has stock to supply the stores with. If the replenishment method for the item-warehouse combination is non-forecasted (Constant, Floating-Point, or Min-Max), the process is the same as Direct to Store. However, if the method does involve forecasting (Dynamic or Time-Supply), then the ROQ for the warehouse is based on the predicted ROQ (or need) of the item (and its associated simple packs) at all stores that are supplied by the warehouse.

In all cases, the ROQ must be distributed over an item's associated simple packs (if any). If the item does not use simple packs, then the issue is moot – the item will be ordered on its own. If there is only one pack associated with the item (referred to as the primary simple pack), then there is no distribution needed – the item will be ordered in this simple pack, since the cost per item for a pack is always less than that of an individual item. If multiple simple packs can be substituted for an item, then the distribution of the ROQ over these packs is determined by comparing the packs' relative sales history. Replenishing an item through multiple simple packs can have a severely negative effect on the performance of this program! Because the pack distribution depends on access to the huge sales history tables (PACKSTORE_HIST, RAG_SKUS_ST_HIST, WIN_STORE_HIST), it is not recommended that many items be placed on replenishment through multiple simple packs. Whenever possible, it is better to assign a primary simple pack to the item, since this does not require distribution calculation.

Once the order quantity for an item (or simple pack) has been calculated, an order line item is written to the ORD_TEMP table if 1) the actual quantity to order is greater than zero, and 2) the replenishment order control indicator for the item-location combination is either Automatic or Semi-Automatic. If it is Manual, a record will be written to another table (REPL_RESULTS) for reporting purposes. If the system-level Replenishment Results indicator is set to "Yes", then all order line items being written to ORD_TEMP will also be written to REPL_RESULTS. If the system-level All Replenishment Results indicator is set to "Yes", all line items will be written to REPL_RESULTS, even if the quantity to order is zero. Crossdock summary records are never written to REPL_RESULTS, since they are only used internally within the replenishment batch process.

The Replenishment Extract module takes into account any substitute items that are set up for the SKUs (staple and fashion). The available inventory of substitute SKUs is used in the calculations that determine the ROQ for stores. If the SKU being replenished has simple packs set up for it, the simple packs are ordered in place of the actual SKU being reviewed for replenishment.

The concept of Due order Processing also occurs in this module with the results written to the repl_results table. An item is considered 'Due' if the accepted stock out level of the item is less than the estimated stock out level for the item (if the item is not ordered today). The stock out level is the allowable stock out quantity for an item. For example, if an item is forecasted to sell 100 units over the items inventory selling days (the amount of days the item is expected to be in stock) and has a service level of 95%, the accepted stock out (ASO) level is 5 units (i.e. up to 5 units can be lost due to stock outs and still be within the items service level objective). The estimated stock out (ESO) level is the amount of stock outs estimated to occur if the item isn't ordered in the current review cycle. For example, if the item has a current available inventory position of 50 units and is forecasted to sell 100 units until the next earliest time it can be reordered and received, the item will experience stock outs. If this level is greater than the accepted stock outs, the item will not achieve its service level and will therefore need to be ordered. Both the ASO and ESO are written with the Replenishment Order Build program using them to determine if the order is Due.

Rplbld

The Replenishment Order Build (rplbld) module builds RMS orders from Recommended Order Quantities (ROQ) generated by the Replenishment Extraction module. These ROQ's are placed on a temporary table (ORD_TEMP) by the Replenishment Extraction module. If contracting is being used in the system, the Contract Process (cntrprss) module associates contracts with the ROQ's on ORD_TEMP created. All records on ORD_TEMP are processed by the Replenishment Order Build module each night. These ORD_TEMP records are placed into logical groups, and a RMS order is created for each logical group.

Depending on the replenishment strategy used for an item location (i.e. Direct to Store, Cross-docked or Warehouse Stocked), different order details are created. The following table breaks down how each stock category is represented in the temporary order table (ORD_TEMP).

Stock Category	Location	Crossdock _store	Crossdock_ind	pack size
Direct to Store (D)	store number	NULL	N	supp_pack_size
Warehouse Stocked (W)	Warehouse number	NULL	N	supp_pack_size
Cross-docked (C) -wh order requirement	Warehouse number	NULL	N	supp_pack_size
Cross-docked (C) - store allocation requirement	Warehouse number	Store number	Y	inner_pack_size

In order to be placed in the same order group, the item/location ROQs from ORD_TEMP must share a common supplier, have the same order status ('W'orksheet or 'A'pproved), and be on the same contract (or not be associated with a contract). Depending on flags on the Order Inventory management (ORD_INV_MGMT) table, two other criteria can be used for splitting order groups. First, if the inventory management level is set to 'D'ept, only items in a single department are allowed in an ordering group. Second, the single location indicator can be set to 'Y'es. If this is the case, only one location is allowed per ordering group. Finally, SKUs may only exist in an ordering group with a single origin country. When an item/loc ROQ ORD_TEMP record is encountered with a different origin country than the one it exists with in the current ordering group, it is placed in a different ordering group.

To assist the recalculation and order scaling processes, the REPL_RESULTS record, associated with the ORD_TEMP being processed, is updated with the order number and allocation number that the ORD_TEMP record was placed with.

The Replenishment Order Build module defaults the appropriate supplier scaling constraints to the orders at the time of creation. If the supplier's inventory management level is set to department, the program will attempt to default the department level constraints to the order. If no constraints exist for that department, the program will attempt to default the supplier level constraints to the order. If no constraints exist for the supplier, constraints will not be assigned to the order.

The Replenishment Order Build module builds orders taking due order processing into account: Due order processing is the idea that creating a large order will be more efficient. The larger order will minimize logistic costs, and may allow vendor minimums to be exceeded. Group ordering considers all items in a vender line when creating orders whether or not all of the items in the vender line require an order to avoid stock outs. If a majority of the items in the vender line will encounter stock outs, the entire vender line will be considered “due”. Vendor lines that are not “due” will still be placed on the order, but they will be flagged as not “due” at the item/location level. It is also possible to set an indicator that says if one item/location is “due” the entire order is due. Orders will be flagged as due at two levels, the order header level, and the order item/location and allocation detail levels. It is up to the buyer reviewing the orders to decide what to do with the order. It may simply be deleted, approved, recalculated, or item/locations that will experience stock outs may be pulled off and put on a separate order (expedite). To facilitate due order processing, ORD_TEMP records will be written for every item reviewed by the Replenishment Extraction module. Due order processing can be turned on/off by an indicator at the supplier/dept level.

Supcnstr

The Replenishment Supplier Constraint Scaling (supcnstr) module scales the purchase orders created in the Replenishment Order Build (rplbld) module. The Replenishment Supplier Constraint Scaling module selects all of the orders created by the replenishment programs which are eligible for scaling. Once selected, the program will serve as a wrapper program and send each order number into the supplier constraint scaling library to actually perform the scaling on the order.

The orders which will be eligible for scaling are as follows:

- If due order processing was used, only orders with a written date of today, origin type = 0 (replenishment order), due order processing indicator = ‘Y’, due order indicator = ‘Y’ and a scale order to constraint indicator = ‘Y’ will be processed. This encompasses all due orders created by replenishment which have constraints associated with them.
- If due order processing was not used, only orders with a written date of today, origin type = 0 (replenishment order), status = ‘A’pproved, due order processing indicator = ‘N’, due order indicator = ‘Y’, and a scale order to constraint indicator = ‘Y’ will be processed. This encompasses all approved orders created by replenishment which have constraints associated with them.

The supplier constraint scaling library scales orders to supplier constraints held on the ORD_INV_MGMT table. Initially one day of supply (based on forecasts if applicable) is added/subtracted to every item/location on the order. The order is then totaled and compared to the scaling constraints. If the order still has not met the constraints, another day of supply is added/subtracted and so forth until the order can be scaled as close as possible to the scaling constraints using days of supply. The solution is then approached even closer by adding/subtracting a pack size (case or pallet) to each item/location on the order. At this level of scaling, order totals are compared to scaling constraints after every item/location until the order quantities can not be scaled any closer to the constraints. At this point, a solution is reached. This method assumes that a days of supply will generally be greater than a supplier pack size. This is generally true for warehouse replenishment or hyper-market store replenishment. For direct to store (other than hyper-market) replenishment, a day of supply would most likely be less than a supplier pack size. The scaling will still perform, except that the solution will not be approached closer while adding/subtracting case sizes, instead the adding/subtracting of days of supply will approach the solution as close as possible.

If a supplier scaling constraint is missed by a significant margin, item order quantities could be significantly inflated to meet the constraint which could result in significant overstock situations at the receiving locations. To reduce the chances of this happening, the user will be able to define maximum additional order scaling quantities at the replenishment item/location level to limit the amount added to item/locations on the order due to scaling. As well, definable at the replenishment item/location level, the user can also set an indicator to prevent an item/location from being scaled at all. If checked, this indicator will exempt the item/location from being scaled.

Depending on whether tolerances have been defined for the scaling constraints, if more than one constraint is used and if item/location scaling constraints are used (see above), scaling solutions may fall within the minimum and maximum defined, within the minimum and maximum defined plus their tolerances or outside of the defined minimum and maximum plus their tolerances. A traffic signal analogy is used in order to determine when the optimal solution is achieved. For example, if the solution is within the defined minimum and maximum, it is considered Green. If the solution is within the defined minimum and maximum plus their tolerances, it is considered Yellow. If the only solution is outside of the defined minimum and maximum plus their tolerances, it is Red. Therefore, if one set up scaling constraints is used, the following scaling solutions exist:

Solution	Constraint 1
1	G
2	Y
3	R

The scaling process will always try to achieve the most optimal solution, which is G (solution 1) in this case. However, due to item/location level constraints, a solution may only be achievable at Y. This solution will still be considered acceptable. If, however, a solution can only be reached at R, the solution will be considered unacceptable by the system. If one has set up 2 scaling constraints, the following solutions would then exist:

Solution	Constraint 1	Constraint 2
1	G	G
2	G	Y
3	Y	G
4	Y	Y
5	G	R
6	Y	R
7	R	G
8	R	Y
9	R	R

Solutions 1 – 4 would be considered acceptable, while solutions 5 – 9 would be considered unacceptable by the system since at least one of the constraints are R (i.e. outside of the defined minimum and maximum including tolerances). If the only solution that can be achieved by the system is unacceptable (i.e. solution 3 with one constraint defined or solutions 5 – 9 with two constraints defined), the order will not be automatically approved during the replenishment process. Whenever an order can not be scaled to meet the constraints, a message will be written out in order to help the user identify why the scaling constraints could not be met. This message can be viewed on-line in the new Order Inventory Management window.

If scaling constraints have been set up as "truckload" constraints where multiple "truckloads" may be required for an order a "truckload" threshold table will be available at the system level. This table would hold truckload threshold information which will determine how rounding should occur at the truckload level. The table will be similar to the following (the actual ranges will be maintained in the VEHICLE_ROUND table):

Lower Range	Upper Range	# of Trucks
0.01	1.20	1
1.21	2.20	2
2.21	3.40	3
3.41	4.40	4

Therefore, if the order quantity initially results in 2.2 truckloads, the number of truckloads would be rounded down to 2. The supplier scaling constraints would then be set as if 2 trucks were being filled. For example, if the maximum supplier constraint was 40,000 lbs, the new super-constraint, given that 2 truckloads are required, would be 80,000 lbs. It should be noted that there is currently no form available to view/maintain this table.

If the replenishment methods being used are not of a forecasting type (i.e. constant, floating point, minimum/maximum), forecasts will not be available to perform days of supply scaling. In this scenario, scaling would be performed a pack size at a time instead of a day at a time.

Rplprg

The Automatic Replenishment Order Maintenance (rplprg) program looks at all approved replenishment orders created during the nightly batch run. These orders are compared with any vendor minimums that may exist. Orders that do not meet the vendor minimums are either deleted or placed in worksheet status. A flag held at the supplier inventory management level determines what action is taken on orders that fail minimums.

Vendor minimums can be held at the order, SKU, or location level. Order and location level minimums are held on the Order Inventory Management table – defaulted from the supplier or supplier/department level during the order creation process in the Replenishment Order Build module. There is a flag that determines if they are applied at the order level or at the location level.

When the order purge indicator on the Supplier Management Inventory record is set to No, a supplier minimum failure at any level causes the order to be placed in worksheet status. When the order purge indicator is set to Yes, a supplier minimum failure at the location level causes the offending location to be deleted, a failure at the SKU level caused the offending SKU to be deleted, and a failure at the order level caused the entire order to be deleted.

For any orders that fail vendor minimums a record is written to the Supplier Minimum Failure (SUPS_MIN_FAIL) table for reporting purposes. This table is purged at the beginning of the program to remove old records.

Any orders that do not fail supplier minimums are then written to the Deals staging table to have any applicable deals, brackets or allowances applied to them. Open to buy is then updated for any orders built in approved status. If any orders are contract orders, the contract amounts are updated as well to reflect any order record deletions.

Additional checks are also performed against orders including:

- If the payment method is letter of credit, the order is placed into worksheet status to allow for the user to enter additional letter of credit information that can not be defaulted.
- If the order's import indicator is set to Yes and the Trade Management module is active, checks are performed to make sure HTS codes are associated to the items that are being imported on the order. If one or more items do not have HTS codes associated to them, the order is set to worksheet status.

This program should run directly after the Replenishment Supplier Constraint Scaling program. It is important that this program runs before any other process affects the generated orders. If the Replenishment Supplier Constraint Scaling module is not being used, it should be directly after the Replenishment Order Build module.

Reqext

Reqext (Item Requisition Extraction) handles automatic replenishment of items from warehouses to stores. It cycles through every item-store combination that is set to be reviewed on the current day, and calculates the quantity of the item that needs to be transferred to the store (if any). Transfers are created and/or records are written to the Replenishment Results table depending on how the order control parameter is set at the item/location level:

- If the order control is set to “Automatic”, a transfer of type Store Requisition (SR) is created in Approved status with a freight code of Normal.
- If it is set to “Semi-Automatic”, a transfer of type Store Requisition (SR) is created in Approved status with a freight code of Hold.
- If the order control is set to “Manual”, replenishment results records are written to the Replenishment Results table against which reports can be run. Using the results from this table, the buyer may then go and manually create a transfer (or perform another transaction) to meet the demand of the location.

The Item Requisition Extract module reviews the item/location as defined within the item/location review cycle (every day, multiple times per week, once every 2 weeks, once every 3 weeks, etc.). The module processes all SKU/store records for fashion/staple SKUs that are being replenished, where the SKU/store is active, its stock category is Warehouse Stocked (W), and its activate date is less than or equal to today and its deactivate date is later than today. In order to be processed, the store must be open.

The module distributes the Recommended Order Quantity (ROQ) over any applicable alternate items associated with the item. The program then takes this information and either creates new transfer line items or adds to existing ones.

Alternate items are either simple packs or substitute items. Simple packs are sellable and orderable packs that contain only a single item, such as a six-pack of cola or twelve-pack of socks. Substitute items are items predefined to be interchangeable with the item being replenished (referred to as the master item).

When an item is set up to use simple packs (designated by an indicator on the REPL_ITEM_LOC table), the ROQ must be distributed among these packs according to desirability. If a master item has no simple packs associated with it, it will be requested as itself. If there is only one pack associated with the item (referred to as the primary simple pack), then there is no distribution needed – the item will be transferred in this simple pack, since the cost per item for a pack is always less than that of an individual item. If multiple simple packs can be substituted for an item, then the distribution of the ROQ over these packs is determined by comparing the packs' relative sales history. Replenishing an item through multiple simple packs can have a severely negative effect on the performance of this program! Because the pack distribution depends on access to the huge sales history tables (PACKSTORE_HIST, RAG_SKUS_ST_HIST, WIN_STORE_HIST), it is not recommended that many items be placed on replenishment through multiple simple packs. Whenever possible, it is better to assign a primary simple pack to the item, since this does not require distribution calculation.

If an item is not set up to use simple packs, the program will see if any substitute items are associated with it. If there are no substitute items associated with the master item, it will be transferred alone. If there are substitute items, they will be fetched into a list and the master item placed at either the head or tail end of the list, depending on the fill priority (set on the SUB_ITEMS_HEAD table). The priority determines which items are transferred first.

No matter what type of alternate items (if any) are used, the program will account for availability when building transfer line items. For simple packs, the share of ROQ allocated to each pack may be decreased or increased if the source warehouse has a shortage of some packs but a surplus of others. For substitute items, transfer quantities are prorated by calculating the ratio of total availability to total need, and items are transferred in order of priority until all need is filled or until no stock is available. If the net inventory is less than the summed ROQ amount, the net inventory will be prorated across all stores. For example:

WH Net Inventory = 200

Store 1 ROQ = 50

Store 2 ROQ = 100

Store 3 ROQ = 200

Total ROQ = 350

Prorated ROQs rounded up to nearest pack size of 3.

Store 1 = $50/350 \times 200 = 30$

Store 2 = $100/350 \times 200 = 60$

Store 3 = $200/350 \times 200 = 117$

Total Prorated ROQ 207

In the above example, the quantity prorated still exceeds the net inventory because of rounding (i.e. $207 > 200$), therefore, store 1 would have its ROQ decreased by one pack size, then store 2, etc. until the total ROQ $\leq$ net inventory. In the above example, store 1 would get 27 units, store 2 would get 57 units, and store 3 would get 114 units.

Once the transfer quantity of an item has been calculated, the transfer line item is posted to the database if 1) the actual quantity to transfer is greater than zero, and 2) the replenishment order control indicator for the item-store combination is either Automatic or Semi-Automatic. If it is Manual, a record will be written to another table (REPL_RESULTS) for reporting purposes. If the system-level All Replenishment Results indicator is set to "Yes", all line items will be written to REPL_RESULTS, even if the quantity to order is zero. Whenever a transfer line item is placed, the appropriate item-location table (RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH and/or PACKWH) is updated to reflect the fact that stock is now reserved for transfer at the warehouse and expected at the store.

Rplrsprg

The replenishment extraction programs (rplext, reqext) write a number of records to the REPL_RESULTS table. This table holds records meant to track certain components of the replenishment algorithms that run during the batch process. Information like order points, safety stock is written to this table for every item/location processed against which the user can perform analysis. Over time, records on this table become unneeded and should be cleared out. The Replenishment Results Purge Program goes through the table and clears out those records that are older than a predetermined number of days.

Module Flow & Scheduling

The Replenishment Attribute Update (rplatupd), Maximum Stock Levels for Fashion/Staple Items (rpladjf/rpladjs), Replenishment Extract (rplext), Replenishment Order Build (rplbld), Replenishment Supplier Constraint Scaling (supcnstr), and the Automatic Replenishment Order Maintenance (rplprg) modules should be scheduled to run nightly.

The Replenishment Supplier Constraint Scaling (supcnstr) module is optional and should only be run if it is desired to have orders scaled to their constraints during the batch cycle.

The Item Requisition Extract (reqext) module should be run nightly in order to maintain optimum stock levels of replenished stock items from the warehouse.

The Automatic Replenishment Order Maintenance (rplprg) module should be scheduled to run directly after the Replenishment Order Build (rplbld) module if scaling is not being done in batch or after the Replenishment Supplier Constraint Scaling (supcnstr) module if scaling is being done in batch. It is important that this program runs before any other non-replenishment process affects the generated orders.

The Replenishment Results Purge (rplsprg) module can be scheduled to run on a daily or as needed basis.

Diagram 1 outlines the programs that are associated with the Replenishment dialog:

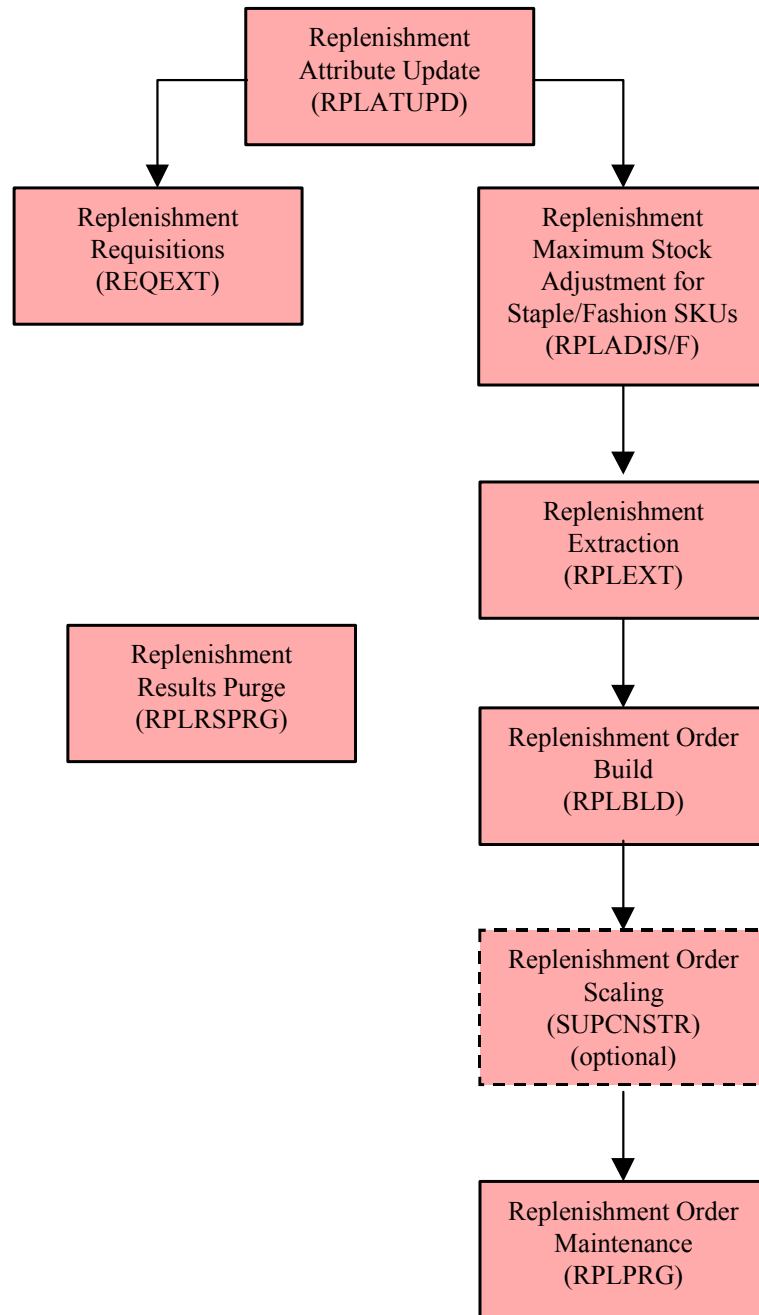


Diagram 1

Retek Forecasting

Functional Overview

The Retek Forecasting batch module consists both of several download modules (i.e. interfacing data from the RMS to Retek Demand Forecasting (RDF)) and upload modules (i.e. interfacing data from the RDF to RMS). The download modules include Staple Merchandise Hierarchy Download (fmednlds), Fashion Merchandise Hierarchy Download (fmednldf), Organizational Hierarchy Download (forgdnld), Warehouse Download (fwhdnld), Staple Weekly Sales Data Download (fsadnlds), Fashion Weekly Sales Data Download (fsadnldf), Staple/Fashion Daily Sales/Issues Data Download (fdaydnld), Staple Weekly Issues (warehouse shipments) Data Download (fsadnlds), Fashion Weekly Issues (warehouse shipments) Data Download (fsadnldf), Item/Store Stockout Download (soutdnld).

The upload programs include Forecasted Weekly Sales/Issues Upload (fcslupld), Forecasted Daily Sales/Issues Upload process, Forecasted Sales Purge (fcstprg) and Forecasted Weekly Sales Rollup (fcstbld).

Note: Some of these programs will create output files for every domain set up in the RMS. A domain is defined as a group of merchandise hierarchies, either grouped by department, class or subclass (ex. Domain 1 consists of departments 1000, 1001, 1002; Domain 2 consists of departments 1003, 1004, 1005, etc.). Due to a file size restriction in RDF, SKUs must be separated into domains with each domain representing a different database instance in RDF to allow for the forecasting of a large number of SKUs. Therefore, for example, the Staple Weekly sales Download (fsadnlds) program will write a record to the Domain 1 output file if the SKU belongs to a department in Domain 1 or the Domain 2 output file if the SKU belongs to a department in Domain 2, etc.

The Staple Merchandise Hierarchy Download (fmednlds) program extracts merchandise hierarchy information for all staple items, whether they are forecastable or not. The merchandise hierarchy information includes the SKU, SKU description, subclass, subclass description, class, class description, department, department description, group, group description, division, division description, primary supplier, supplier description, the SKUs forecast indicator which determines if the SKU is forecastable or not and the domain in which the SKU resides. In order to keep the fashion and staple output files consistent, the SKU description is also used to populate the style description field in the output file. The style ID field is left blank.

The Fashion Merchandise Hierarchy Download (fmednldf) program extracts merchandise hierarchy information for all fashion SKUs, whether they are forecastable or not. The merchandise hierarchy information includes the SKU, SKU description, style, style description, color, color description, size 1, size 1 description, size 2, size 2 description, subclass, subclass description, class, class description, department, department description, group, group description, division, division description, primary supplier, supplier description, the SKUs forecast indicator which determines if the SKU is forecastable or not (this indicator is set at the style level, therefore, if the style is forecastable all SKUs within the style are forecastable) and the domain in which the SKU resides (again, this is determined at the style level).

The Organizational Hierarchy Download (forgdnld) program extracts organizational hierarchy information for all stores set up in the RMS. The organizational hierarchy information includes the store, store name, the district to which the store is associated, district name, region, region name, area, area name, chain, chain name, company, company name, store format, store format description. An additional output file includes the store ID, the store's open and close date and the store's selling square footage (or square meters, depending on the measurement type system option field). A set of output files is created for each domain within Retek. Since the organizational hierarchy is independent of the domain, each output file for each domain will be the same, and will contain all stores and their information.

The Warehouse Download (fwhdnld) program extracts warehouse information for all warehouses set up in the RMS. The information includes the warehouse ID and warehouse name. An output file is created for each domain within Retek. Since warehouses are independent of the domain, each output file for each domain will be the same, and will contain all warehouse.

The Staple Weekly Sales Data Download (fsadnlds) program extracts staple SKU sales history information (located on the WIN_STORE_HIST table) at the store/week level for all staple SKUs set up as forecastable. The last sales export date on the staple SKU/store table (WIN_STORE) drives the weeks for which sales information will be extracted for the SKU/store from the sales history table. All weeks greater than the last sales export date and not greater than the last week will be extracted. If the last sales export date is blank, this indicates that all weeks are to be downloaded (ex. when a SKU changes domains or when a SKU is set to forecastable for the first time). As well, if the domain to which the SKU is associated has been flagged to download all sales information, all sales history for the SKU will be downloaded (the load sales indicator flag is located on the DOMAIN_DEPT, DOMAIN_CLASS and DOMAIN_SUBCLASS tables). This flag may be set to Yes when a merchandise hierarchy changes domains or is first associated to a domain. Three output files are created for each domain. One file for regular sales, one for promotional sales and one for clearance sales.

The Fashion Sales Data Download (fsadnldf) program extracts fashion SKU sales history information (located on the RAG_SKUS_ST_HIST table) at the store/week level for all fashion SKUs whose associated style is set up as forecastable. The last sales export date on the fashion SKU/store table (RAG_SKUS_ST) drives the weeks for which sales information will be extracted for the SKU/store from the sales history table. All weeks greater than the last sales export date and not greater than the last week will be extracted. If the last sales export date is blank, this indicates that all weeks are to be downloaded (ex. when a SKU changes domains or when a SKU is set to forecastable for the first time). As well, if the domain to which the SKU is associated has been flagged to download all sales information, all sales history for the SKU will be downloaded (the load sales indicator flag is located on the DOMAIN_DEPT, DOMAIN_CLASS and DOMAIN_SUBCLASS tables). This flag may be set to Yes when a merchandise hierarchy changes domains or is first associated to a domain. Three output files are created for each domain. One file for regular sales, one for promotional sales and one for clearance sales.

The Staple/Fashion Daily Sales/Issues Data Download (fdaydnld) program extracts fashion and staple SKU daily sales history information and daily issues information (located on the TRAN_DATA_HIST table) at the location/day level for all fashion and staple SKUs set up as forecastable. The last sales export date on the item/store tables (WIN_STORE, RAG_SKUS_ST) and the last issues export date on the item/warehouse tables (WIN_WH, RAG_SKUS_WH) drives the weeks for which daily sales/issues information will be extracted for the SKU/store or SKU/warehouse from the transaction data history table. All weeks greater than the last sales/issues export date and not greater than the last week will be extracted. If the last sales/issues export date is blank, this indicates that all weeks are to be downloaded. As well, if the domain to which the SKU is associated has been flagged to download all sales/issues information, all sales and issues history for the SKU will be downloaded. Three files for SKU/stores are created; one file for regular sales, one for promotional sales and one for clearance sales. One file for SKU/warehouses is created.

The Staple Weekly Issues Data Download (fisdnlld) program extracts staple SKU issues (i.e. warehouse shipments) history information (located on the WIN_WH_HIST table) at the warehouse/week level for all staple SKUs set up as forecastable. The last issues export date on the staple SKU/warehouse table (WIN_WH) drives the weeks for which issues information will be extracted for the SKU/warehouse from the issues history table. All weeks greater than the last issues export date and not greater than the last week will be extracted. If the last issues export date is blank, this indicates that all weeks are to be downloaded (ex. when a SKU changes domains or when a SKU is set to forecastable for the first time). As well, if the domain to which the SKU is associated has been flagged to download all issues information, all issues history for the SKU will be downloaded (the load issues indicator flag is located on the DOMAIN_DEPT, DOMAIN_CLASS and DOMAIN_SUBCLASS tables). This flag may be set to Yes when a merchandise hierarchy changes domains or is first associated to a domain. One output file is created for each domain.

The Fashion Issues Data Download (fisdndf) program extracts fashion SKU issues history information (located on the RAG_SKUS_WH_HIST table) at the warehouse/week level for all fashion SKUs whose associated style is set up as forecastable. The last issues export date on the fashion SKU/warehouse table (RAG_SKUS_WH) drives the weeks for which issues information will be extracted for the SKU/warehouse from the issues history table. All weeks greater than the last issues export date and not greater than the last week will be extracted. If the last issues export date is blank, this indicates that all weeks are to be downloaded (ex. when a SKU changes domains or when a SKU is set to forecastable for the first time). As well, if the domain to which the SKU is associated has been flagged to download all issues information, all issues history for the SKU will be downloaded (the load issues indicator flag is located on the DOMAIN_DEPT, DOMAIN_CLASS and DOMAIN_SUBCLASS tables). This flag may be set to Yes when a merchandise hierarchy changes domains or is first associated to a domain. Three output files are created for each domain. One file for regular issues, one for promotional issues and one for clearance issues.

The Item/Store Stockout Download (soutndf) program extracts all staple and fashion SKU/stores that have a stock position of zero or less than zero (this information is found on the WIN_STORE and RAG_SKUS_ST tables). Only forecastable SKUs are considered.

The Forecasted Sales Purge program (fcstrprg) purges all of the forecasted data for a specified domain. When the program is executed, a domain is specified and all of the forecast tables are truncated for the specified domain's data (ITEM_FORECAST, DAILY_ITEM_FORECAST, SUBCLASS_SALES_FORECAST, CLASS_SALES_FORECAST, DEPT_SALES_FORECAST). This program should be executed before the Forecasted Sales Upload (fcslupld) and Forecast Sales Rollup (fcstrbld) programs are run, as these programs will insert new forecasted sales/issues data into these tables.

The Forecasted Weekly Sales/Issues Upload (fcslupld) program reads in the RDF forecasted sales/issues output file (forecasted sales at the SKU/store/week level and issues at the SKU/warehouse/week level) and writes the information to another file formatted for the RMS to upload the information via SQL loader into the forecasting item table (ITEM_FORECAST). There will be a forecasted sales/issues file from the RDF for each domain. Along with the forecasted sales/issues, a confidence factor is also included with each forecast. As each record is read, if the record passes all validation it is written to the SQL loader file. Valid records are those with valid end of week dates that are greater than or equal to the current end of week date. As well, a temporary table (FORECAST_REBUILD) is populated with the domains which are receiving new forecasts to indicate that item level data of these domains should be rolled up to the subclass, class, and department level (SUBCLASS_SALES_FORECAST, CLASS_SALES_FORECAST, DEPT_SALES_FORECAST) by the Forecast Sales Rollup program (fcstrbld).

The Daily Forecasted Sales/Issues Upload process reads in the RDF forecasted daily sales/issues output file (forecasted sales at the SKU/store/day level and issues at the SKU/warehouse/day level). The RDF output file is SQL loaded straight into the daily forecasting item table (ITEM_FORECAST). No intermediary program is used to reformat the RDF output file into an RMS format like with the weekly sales/issues upload program. There will be a forecasted sales/issues file from the RDF for each domain. Along with the forecasted sales/issues, a confidence factor is also included with each forecast.

The Forecast Sales Rollup (fstrbld) program rolls up the forecasted sales information from the SKU/store/week level (located ITEM_FORECAST table) to the subclass/store/week level (located on the SUBCLASS_SALES_FORECAST table), class/store/week level (located on the CLASS_SALES_FORECAST table) and the department/store/week level (located on the DEPT_SALES_FORECAST table). The forecast build table (FORECAST_REBUILD) drives which domains are to be rolled up. This table is populated by the Forecasted Weekly Sales/Issues Upload module as forecasted sales/issues information for SKUs are read in. This program will only roll up sales forecasts and not issues forecasts as well as only weekly sales, not daily sales.

Module Flow and Scheduling

The Forecasted Sales Purge (fcstrprg), Forecasted Weekly Sales/Issues Upload (fcsupld), Forecasted Daily Sales/Issues Upload process programs should be run in Phase 2 of the batch cycle. These programs can be run nightly or whenever forecasts need to be interfaced from RDF. The Forecasted Sales Rollup (fstrbld) program runs in Phase 3 and should be run whenever the Forecasted Weekly Sales/Issues Upload program runs.

The Staple Merchandise Hierarchy Download (fmednlds), Fashion Merchandise Hierarchy Download (fmednldf), Organizational Hierarchy Download (forgdnld), Warehouse Download (fwhdnld), Staple Weekly Sales Data Download (fsadnlds), Fashion Weekly Sales Data Download (fsadnldf), Staple/Fashion Daily Sales/Issues Data Download (fdaydnld), Staple Weekly Issues (warehouse shipments) Data Download (fsadnlds), Fashion Weekly Issues (warehouse shipments) Data Download (fsadnldf), Item/Store Stockout Download (soutdnld) programs should be run in Phase 4 of the batch cycle. Although the programs can be run throughout the week, it is generally recommended that the extracts are run on a weekly basis.

The following diagram 1 outlines the individual programs that are associated with the Retek Forecasting batch dialog, download and uploads:

Downloads

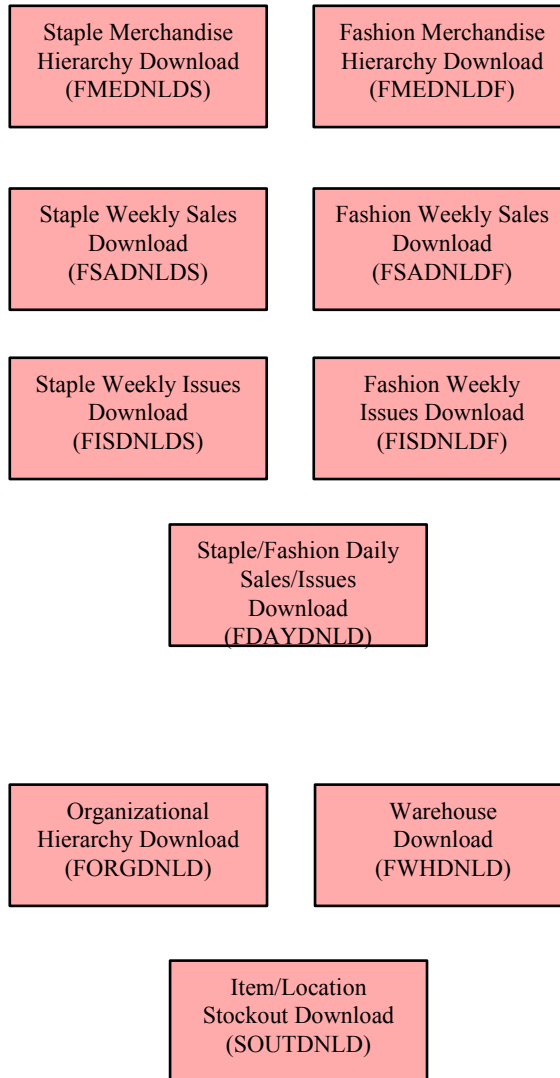
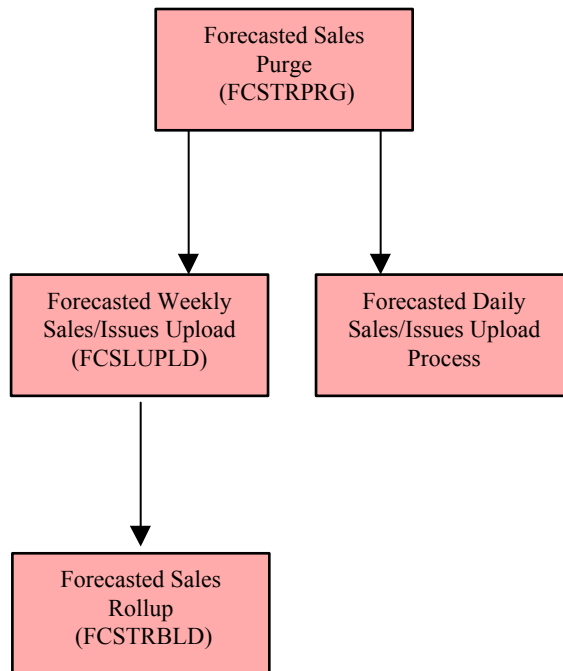


Diagram 1

Uploads



Return to Vendor

Functional Overview

The Return to Vendor (RTV) module performs two major functions:

- Accepts RTV transactions from an external source and processes each transaction accordingly.
- Purges old RTV information from the appropriate tables.

The RTV Transaction Upload (rtvupld) module processes RTV transactions captured by an external source according to the same logic as the online RTV functionality within the RMS. For each RTV transaction processed by this module, an RTV is created in Shipped status on the RMS database. Additionally, appropriate updates are made to the item's perpetual inventory, average cost, cycle count on hand snapshots, and unavailable inventory.

If the RMS system is interfacing with the Retek Distribution Management (RDM) system, then the RTV Transaction Upload Conversion (lifrtvup) module must run before the rtvupld module. The conversion module will reformat the RDM inventory adjustment file into a format that is acceptable to the rtvupld module.

The RTV Purge (rtvpgr) module deletes out-dated RTV records from the system. RTV transactions that have been completed shipped or cancelled and those for which the completion date exceeds the RTV retention period are deleted. The RTV retention period is maintained on-line as a system option.

Module Flow and Scheduling

The RTV Transaction Upload (rtvupld) module is designed to be executed daily; however, it can be executed multiple times throughout the day if more timely RTV information is required. This module is usually scheduled to execute with the other inventory transaction upload modules (i.e. receiving, inventory adjustments, transfers, etc.).

If interfacing with RDM, the lifrtvup module must run immediately before the rtvupld module.

The RTV Purge (rtvpgr) module can be scheduled to run at any time; however, practical usage is to run it only once at the end of each month at the end of all batch cycles.

The following diagram 1 outlines the modules that are associated with the Return to Vendor dialog:

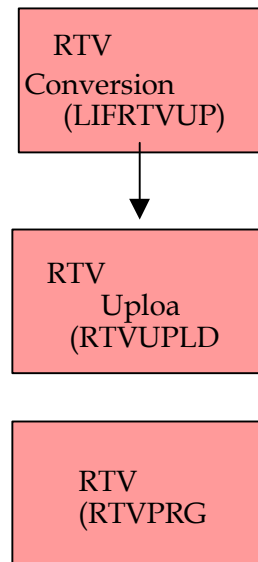


Diagram 1

Sales Posting

Functional Overview

The Sales Posting batch component is executed in order to process sales data brought in from an external system while also maintaining sales history within the Retek system. The POS Upload (posupld) module is run on a daily basis at the beginning of the batch processing. The purpose of this batch module is to process sales and return details from an external point of sale system. Sales history, using Sales History Rollup by Department, Class and Subclass (hstbld) module, is then collected for each SKU in the system. The collected history information is stored at the subclass, class, and department level. For each SKU, the data to be saved includes sales quantity, value, gross profit, and sales rate. The Sales History Rollup by Department, Class and Subclass (hstbld) module should be run with the parameter Weekly. After sales history is built, the Pre-Post module is run in order to maintain the rebuild tables.

Four batch modules are run weekly in order to update the current stock on hand, retail values, and average cost for each item. The modules, including Weekly Stock on Hand and Retail Value Update for Staple SKU/Store (hstwsst) module, Weekly Stock on Hand and Retail Value for Staple SKU/Warehouse (hstwswh) module are run in order to process data for SKUs at stores and warehouses. Weekly Stock on Hand and Retail Value Update for Fashion SKU/Store (hstwkfst) module, Weekly Stock on Hand and Retail Value Update for Fashion SKU/Warehouse (hstwkfwh) module are run in order to process styles at stores and warehouses.

The Purge Sales History (hstprg) module is run ad hoc in order to maintain the style/SKU location history tables. Any history more than a specified number of months old is deleted. Number of months for retention of fashion style history is specified by the system's options.

Module Flow and Scheduling

The POS Upload (posupld) module should be run on a daily basis as point of sales information becomes available in the system. The Sales History Rollup by Department, Class and Subclass (hstbld) module is run with the input parameter Weekly in order to rebuild sales at the weekly time frame. A pre-post is performed after this.

The Weekly Stock on Hand and Retail Value Update for Staple SKU/Store (hstwsst) module, Weekly Stock on Hand and Retail Value for Staple SKU/Warehouse (hstwswh) module, Weekly Stock on Hand and Retail Value Update for Fashion SKU/Store (hstwkfst) module, Weekly Stock on Hand and Retail Value Update for Fashion SKU/Warehouse (hstwkfwh) module should be scheduled on the last day of the week after replenishment and transfers have been completed.

The Purge Sales History (hstprg) module should be run for maintenance purposes. Practically, it should be run at the end of the month after all other batch processing is completed.

The following diagram outlines the individual modules that are associated with the Sales Posting dialog:

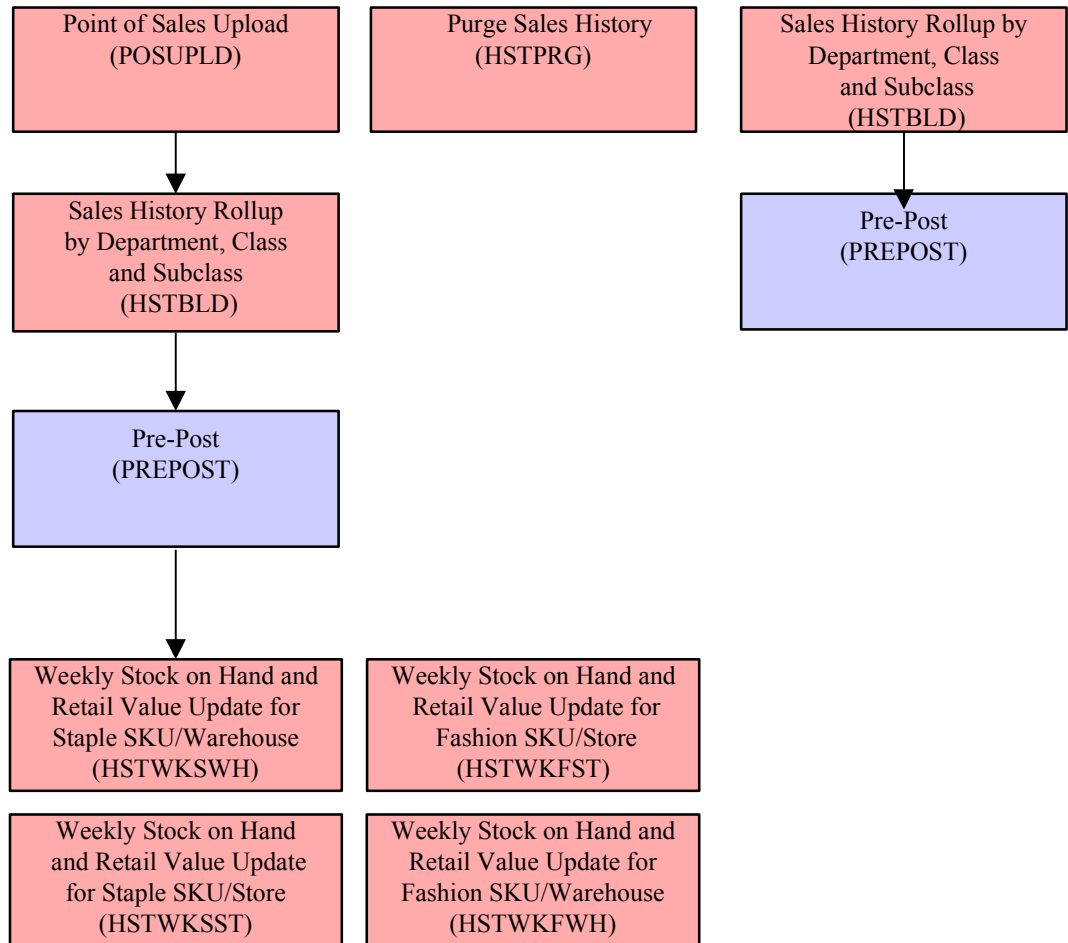


Diagram 1

Stock Counts

Functional Overview

Within the RMS, stock count processing falls into two distinct categories-Unit and Dollar or Unit Only.

Stock Count - Unit and Dollar

The department, class, or subclass is used when requesting a unit and dollar stock count because the results of the count are fed into the stock ledger, which is maintained at the subclass level. As a result of this type of stock count, the stock on hand is adjusted by the physical count units, the stock ledger is adjusted by physical count dollars, and the shrinkage dollars/percent is calculated at subclass/location level.

Stock Count - Unit Only

An Item List is used when requesting a unit only stock count so that a user can group items together as required. As a result of this type of stock count, the stock on hand is adjusted by the physical count of units. The stock ledger is not adjusted and the shrinkage dollars/percent are not calculated.

The method used in the stock count processing relies on a snapshot of stock on hand taken for the stock count location at the close of the business day. This location must complete the counting and recording process before the location is reopened for business the next day. A stock on hand adjustment is then created by comparing the snapshot with the physical count of units as it becomes available. As a result, the normal day-to-day selling, receiving, and transfer of stock can continue uninterrupted once the counts have been captured.

A snapshot of the in-transit quantity, unit retail, and weighted average cost are also taken at the same time the snapshot of stock on hand is taken.

Late transactions coming from the Point of Sale (i.e. sales), the Warehouse Management System (i.e. transfers, RTVs, receipts, etc.) adjust the snapshot stock on hand, so long as the physical count units have not been used to adjust the stock on hand yet.

Reports of variances between book stock and the counts can be produced, along with in-transit quantity. Changes to the counts can be entered online in cases where discrepancies are determined to be the result of errors in counting or stock identification, items supposedly in transit being counted, or other causes.

Once physical counts are finalized, users can create unit adjustments to bring the book stock on hand into line with the physical counts.

For a Unit and Dollar stock count, the final stock count units, along with snapshots of in-transit quantity, unit retail, and unit average cost, are used to generate total dollar figures for every subclass/location included in the stock count. These dollar figures can be reviewed and adjusted if necessary before submitting for stock ledger processing. Every subclass/location included in a Unit and Dollar stock count must be processed before the month-end stock ledger process (batch job salmth) can be run.

For Unit and Dollar type stock counts, if users decide not to completely process either a location or an entire stock count, they must delete the location(s) or the entire stock count. The Dollars View screen or Missing Stock Count Result Report can be used to find out which locations have not been processed. Any unprocessed subclass/locations on a stock count will cause month-end stock ledger processing to stop. However, once stock on hand has been adjusted by the physical counts, the stock count or the location cannot be deleted and must be processed to completion.

A Shrinkage Report can be run to view the book stock dollars, stock count dollars, and shrinkage results.

The batch processing for Stock Counts works as follows: After a stock count has been requested within the RMS, the Stock Count Explode to SKU/Locations for Stores (stkxplst) module and Stock Count Explode to SKU/Locations for Warehouses (stkxplwh) module are used to explode 'unit & dollar' type stock count records from the subclass/class/dept/location level to the item/location level. This program should run daily after the sales audit process is complete. Both modules are date driven and process stock counts which are to occur within the number of days specified by the system variable lockout days.

The daily scheduled Stock Count Snapshots Update (stkupd) module is also date driven. It takes snapshots of stock on hand for each item/location record on the day of the cycle count. This program must run daily after the sales audit process is complete, and after the Stock Count Explode to SKU/Locations programs have run.

Actual counts from a stock count are uploaded into the RMS by the Upload Stock Count Results (stkupld) module. This program will be run as needed with the other external interface programs. If interfacing with the Retek Warehouse Management System, then the Stock Count Upload Conversion (lifestkup) module must run before the Upload Stock Count Results module. The conversion module will reformat the Nautilus inventory balance file into a format that is acceptable to the Upload Stock Count Results module.

The Stock Count Stock on Hand Updates (stkvar) module runs initially to update stock on hand and budget information based on stock count results. It will update stock on hand information on the Item/Location tables, and if the cycle count adjustment is both unit and dollar, it will write stock ledger overlay information to the STAKE_PROD_LOC table. This information will then be used in the Stock Count Shrinkage Update module to update the stock ledger tables. The Stock Count Stock on Hand Updates program should run daily before the external interface programs which check stock on hand during processing, as well as before the Stock Count Shrinkage Update program.

The Stock Count Shrinkage Update (stkdlly) module is then run to update the weekly data, monthly data, and half data tables based on stock count data written to the STAKE_PROD_LOC table by the Stock Count Stock on Hand Updates module. It should run after the Stock Count Stock on Hand Updates program, and before the Stock Ledger Monthly program which calculates the month data closing stock position with the actual shrinkage amounts (see the Stock Ledger section for more details about this program).

The Purge Stock Count (stkprg) module deletes records from the cycle count tables where the count is more than 30 days old.

Module Flow and Scheduling

The Stock Count Explode to SKU/Location for Stores (stkxplst) module, Stock Count Explode to SKU/Location for Warehouses (stkxplwh) module, Stock Count Snapshots Update (stkupd) module, Stock Count Stock on Hand Updates (stkvar) module should be run in Phase 3 of the batch schedule.

The Upload Stock Count Results (stkupld) module, Stock Count Upload Conversion (lifestkup) module – if interfacing with the Retek Warehouse Management System should be run in Phase 1 of the batch schedule.

The Purge Stock Count (stkprg) module can be scheduled to run at any time. However, practical usage is to run it only once at the end of each month after all other batch cycles.

The following diagram 1 outlines the individual programs that are associated with the Stock Counts dialog and are listed in the order in which a stock count would be processed by them:

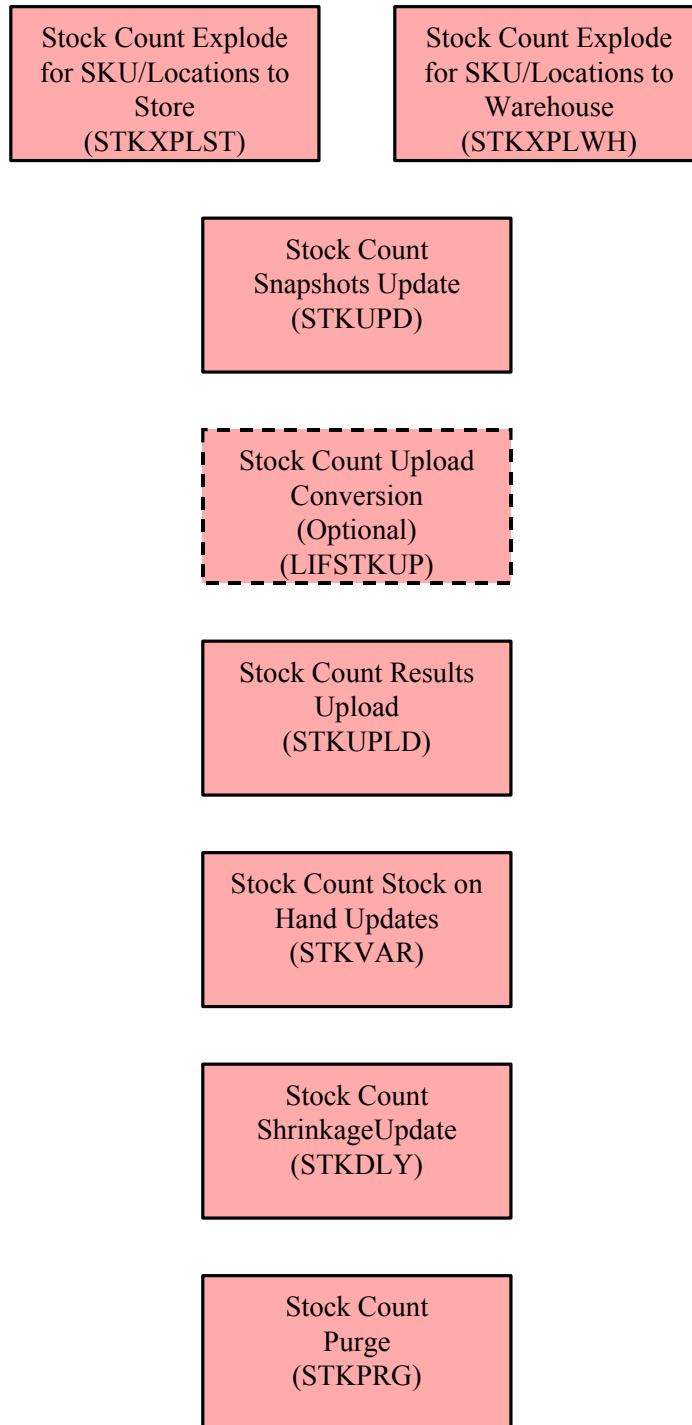


Diagram 1

Stock Ledger

Functional Overview

Stock ledger is implemented at subclass/location level in the Retek base product, which means inventory and gross margin are calculated and stored on WEEK_DATA and MONTH_DATA tables at subclass/location level. Retek supports both retail and cost method of accounting, defined at the department level (i.e. profit calculation type). If retail accounting method is chosen, cumulative merchandise statistics, i.e. Goods Available For Sale (GAFS) is kept on a seasonal (or half) basis, i.e. reset at the beginning of each half. If a stock count has occurred during a week or month, the Stock Count batch module stkdly will update WEEK_DATA and MONTH_DATA with the actual shrinkage calculated from stock count results. This actual shrinkage will be used to adjust the inventory when weekly and monthly stock ledger processing modules are run. If a retailer chooses to reduce each period's book inventory by budgeted (or estimated) shrinkage amount (by turning on system option: budget shrinkage indicator), budgeted shrinkage percent stored on HALF_DATA_BUDGET will be used to multiply the sales amount for the period to derive the budgeted shrinkage amount.

Stock Ledger transaction data (i.e. sales, purchases, receipts, etc.) is stored on TRAN_DATA table at sku/location level. Each day, after all transactions have been written to TRAN_DATA, the sales stage program (salstage) moves all of the transactions to IF_TRAN_DATA. Salstage first truncates the IF_TRAN_DATA table. It then moves every record on TRAN_DATA to IF_TRAN_DATA. Lastly it truncates TRAN_DATA.

The Daily Stock Ledger Processing (saldly) module rolls up transaction data on IF_TRAN_DATA into subclass/location level on DAILY_DATA. All systems interfacing with RMS also perform their extracts from IF_TRAN_DATA. When saldly and the interface modules have completed, the sales append program (salapnd) appends all of the stock ledger data on IF_TRAN_DATA onto the TRAN_DATA_HISTORY table.

The Weekly Stock Ledger Processing (salweek) module rolls up data on DAILY_DATA to subclass/location/week level on WEEK_DATA. This process should run at the end of each week, with no requirement that all the transaction data be collected for that week. This is because salweek accepts 'late' transactions, which are transactions for the previous weeks, and updates the appropriate week, as long as the transaction date belongs to a month that has not been 'closed' yet, i.e. it updates all weeks since the last 'closed' month. Months are closed by running salmth. The pre-post module for salweek should be run after each weekly run of salweek to update the appropriate system variables. Salweek only runs one week of data at a time. In addition, salweek should be run immediately prior to running the monthly process to ensure that WEEK_DATA is in sync with MONTH_DATA (pre-post module need not run in this case). Note that if a retailer uses regular calendar (vs. 454 calendar) for their stock ledger, inventory and gross margin will not be available on a weekly basis (i.e. salweek will not be run).

The Monthly Stock Ledger Processing (salmth) module rolls up data on DAILY_DATA to subclass/location/month level on MONTH_DATA. If a stocktake has been done during the month, all stocktake results must have been processed. The monthly process requires that all transaction data for that month be accounted for before it can run, because once monthly process is completed, that month is considered 'closed', and 'late' transactions will NOT be processed into a 'closed' month automatically. It is strongly recommended that user establish procedures that ensure all data for that month be collected before running the monthly process. The pre-post module should be run afterwards to update the appropriate system variables.

In the case where for some reason, a retailer must run the monthly process a set number of days after the end-of-month-date, i.e. monthly process can not wait until all transactions are accounted for, and if large volume of late transactions need to be processed on regular basis, it would be up to the retailer to define how they want to handle late transactions.

The Stock Ledger and Budget Tables Insert (salins) module is used to populate the stock ledger and budget tables: WEEK_DATA, MONTH_DATA, HALF_DATA, MONTH_DATA_BUDGET, and HALF_DATA_BUDGET for each subclass/location or department/location combination in the system at installation time. Subsequently, this module should be run daily to pre-insert a new record into the stock ledger and budget tables whenever a new location, department or subclass is added to the system.

The End-of-Half Stock Ledger Processing (saleoh) module will purge rows on DAILY_DATA, WEEK_DATA, MONTH_DATA, HALF_DATA, MONTH_DATA_BUDGET and HALF_DATA_BUDGET that are 18 months or older. It inserts 6 rows of MONTH_DATA_BUDGET and one row of HALF_DATA_BUDGET for next year for each department/location. It also rolls up this year's HALF_DATA.shrinkage_pct at subclass/location level to HALF_DATA_BUDGET.shrinkage_pct at department/location level for next year.

Transactions that are older than a specified number of retention days are purged using the Purge Stock Ledger Transactions (salprg) module. The transaction data retention days is defined as system level option.

Stock ledger modules may work differently if the system VAT indicator is on or off.

Module Flow and Scheduling

The Stock Ledger and Budget Tables Insert (salsins) module and Daily Stock Ledger Processing (saldly) module should be scheduled to run on a daily basis.

The Weekly Stock Ledger Processing (salweek) module should be scheduled to run at the end of the week, after saldly. The Pre-Post module should follow this module.

Monthly, the Weekly Stock Ledger Processing (salweek) module and Monthly Stock Ledger Processing (salmth) module should be scheduled to run with the Pre-Post module for and Monthly Stock Ledger Processing (salmth) module following.

The End of Half Stock Ledger Processing (saleoh) module should be scheduled to run at the end of the half after the monthly process has been completed for month 6 of the current half and before the monthly process for month 1 of next half.

The Purge Stock Ledger Transactions (salprg) module can be run at any point, but preferably should be run outside of the batch window to reduce contention.

The following diagram 1 outlines the individual modules that are associated with the Stock Ledger dialog:

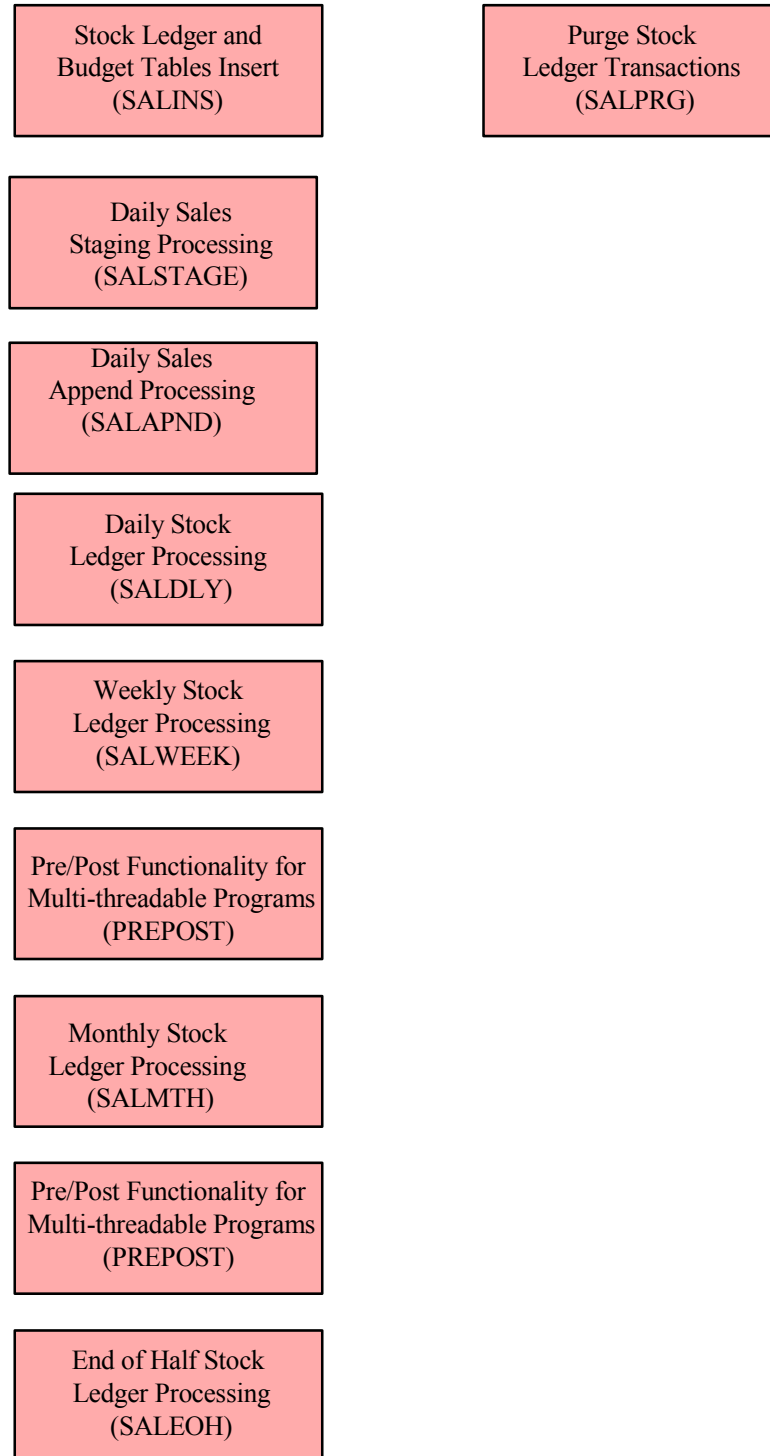


Diagram 1

Supplier Maintenance

Functional Overview

The Supplier Data Amount Repository (supmth) module is executed based on multiple transaction types for each department/supplier combination in the system. Its primary function is to convert daily transaction data to monthly data. After all of the data is converted, the daily information is deleted to reset the system for the next period (supmth_post).

The Supplier Download (supdnld) module will download supplier information, such as address and quality control percent sampling, to the warehouse management system.

Module Flow and Scheduling

The Supplier Data Amount Repository (supmth) module should be run during Phase 3 of the batch process on a monthly basis.

The Supplier Download (supdnld) module can be run at any time during the batch schedule, however, it is recommended to run during Phase 1.

The following diagram 1 is the individual program that is associated with the Supplier Maintenance dialog:

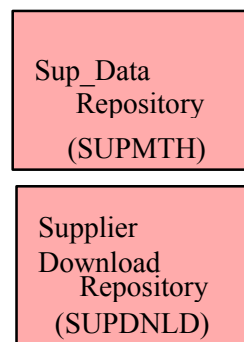


Diagram 1

System Administration and Control Files

Functional Overview

The Pre-and Post-Processing Functionality for Multi-threadable programs (prepost) modules are responsible for executing general system administration functions either before or after certain programs are run.

The Daily Record Deletion (dlyprg) module is executed nightly to delete all of the records in the system that have been marked for deletion by the online system during the day. Before deletion, all relations are checked to ensure that the record may be deleted. For example, if a staple SKU has been marked for deletion, this module will check that the SKU has not been placed on order later in the day. If relations are found to exist, they are noted on a report that will itemize any problems found when running the module, and the record will not be deleted that night.

Module Flow and Scheduling

The Pre-and Post-Processing Functionality for Multi-threadable programs (prepost) module should be run before or after programs requiring specific processing to run or complete successfully.

Tickets/Labels

Functional Overview

The Tickets/Labels batch component creates interface file to an external ticket printing system. Ticket Download, the only module for this process, is run to create an output file containing all of the information to be printed on a ticket or label for a particular SKU and location. Information to be printed on the ticket is retrieved from the RMS, based on the SKU, location, and the ticket type requested.

Module Flow and Scheduling

Ticket download is scheduled to run daily, during any phase.

The following diagram 1 outlines the individual programs that are associated with the Tickets/Labels dialog:

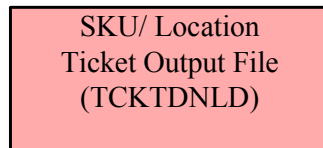


Diagram 1

Transfers

Functional Overview

Within the RMS, there are ten types of transfers.

The first three types: Store Requisition (SR), Cross-dock PO (PO), and Stock Allocation (SA) are for the purpose of distributing stock from warehouses to other locations, i.e. warehouse as source location, and either store or other warehouse as destination location. They are also called 'transfer requests', since they are meant to be combined into the fourth type, Combined Transfer (CT) (with one exception as explained below for SA type of transfer) and then either deleted from the system or its status updated to indicate that the transfer has reached the end of its life cycle. There are two forms created just for maintaining transfer requests : transfer requests by location (tsfbystl) and transfer requests by item (tsfbysku). Combined Transfer (CT) is used to consolidate the above 'transfer requests' to create one manifest per source/destination location combination. Combined transfer is the transfer that will be shipped and received with.

The next four types of transfers: Return to Vendor (RV), Confirmation (CF), Administrative (AD) and Manual Requisition (MR), although are created as different types, will be processed exactly the same way as CT type of transfer, i.e. the processing logic for these types of transfers is exactly the same within RMS, as far as shipping and receiving them are concerned.

The last two types of transfers: Customer Order (CO) and Non-salable Merchandise (NS) are processed a little differently as explained below.

The following are detailed descriptions of these ten types of transfers:

SR

Store Requisition - Created as result of warehouse to store replenishment batch process. This type of transfer is deleted from the system after it is combined into CT type of transfer.

PO

Cross-dock PO - Created as result of receiving a cross-dock non-premark Purchase Order with allocation attached. This type of transfer is deleted from the system after it is combined into CT type of transfer. CT type of transfers are created for Premark Purchase Orders, since these are received via carton and can't be combined with other transfers.

SA

Stock Allocation – RMS provides the capability of creating transfers from one location to many locations as user allocates an item or item list across many locations. Transfers created in this fashion are the ‘Stock Allocation’ type of transfer. There is a separate on-line dialog for creating this type of transfers. Multiple transfer header records of ‘SA’ type are created for a single allocation, which are tied together with an allocation number, and can be viewed and maintained together in one dialog. User may choose to allow this type of transfer to be combined into CT type of transfer by setting merge indicator to ‘Y’, in which case, the status of the transfer will be updated to ‘M’ (Released for Shipment or merged) after it is combined, and the transfer essentially has reached the end of its life cycle. Or user may set merge indicator to ‘N’ so that the status of this transfer can be monitored from the time it is created until it is shipped and received.

CT

Combined Transfer - Transfer created in Combine Warehouse Originated Transfers (tsfcomb) module, which combines the transfer requests (i.e. SR, PO, and SA transfer type (with merge_ind = ‘Y’)) with the same source and destination location into a single transfer. After a SR or PO type of transfer is combined into a CT type of transfer, it is deleted from the system, while SA type of transfer (with merge_ind = ‘Y’) is updated with status = ‘M’ (released for shipment or merged)

RV

RTV – A transfer type to indicate that merchandise is transferred for the purpose of eventually returning the merchandise to the vendor.

CF

Confirmation – A transfer type to indicate that it has already occurred.

AD

Administrative – A transfer type to indicate its administrative (vs. merchandising) purpose and thus allows different interfaces created to an external system, e.g., for purpose of charging a different expense account in the General Ledger

MR

Manual Requisitions – A general-purpose transfer that does not fall into any of the other categories. For example, a store to store transfer could be created using this type of transfer.

CO

Customer Order - A transfer created for a specific customer. A customer record is created for this type of transfer, which includes customer's name, address and the delivery type : customer pick up or ship direct. If the delivery type is ship direct, the Transfer Shipments Upload (tsfoupd) module will perform all the logic to close or receive the transfer as it ships the transfer.

NS

Non-Salable Merchandise - Moves stock directly from an unavailable inventory bucket at source location to the corresponding unavailable inventory bucket at destination location

The transfer batch processing consists of the following modules: The first four have to do with shipping and receiving transfers:

- Transfer Shipments Upload (tsfoupd)
- Transfer Shipments Upload Conversion (lifbolup)
- P.O. Receipt, Transfer Receipt, and Carton Receipt Parse (tsfparse)
- Transfer Receipts Upload (tsfiupd)
- Carton Receiving Upload for P.O. & Transfers (ctniupd)

The following three are for reserving stock, combining transfers and creating interface file:

- Reserve Inventory for Stock Allocation (tsfresv)
- Combine Warehouse Originated Transfers (tsfcomb)
- Approved Warehouse Transfers Download (tsfdnld)
- Transfers destined to a Warehouse (tsfdcdld)

The following two are for purging transfer records :

- Transfer Record Purge (tsfpgr)
- Stock Allocation Transfer Record Purge (tsfalprg)

A high level functional description of these modules are as follows:

The shipment transactions relating to transfer that are input from an external system are processed by the tsfoupd module. The transfer shipment transactions fed into the system will update existing transfers and shipments, or create new transfers and shipments, update perpetual inventory for source locations, and create stock ledger transactions, etc. If interfacing with the Retek Distribution Management system (RDM), then a conversion program (lifbolup) must first convert the warehouse Bill of Lading file into a file that is acceptable by the tsfoupd module.

The receipt transaction file created from an external source like WMS or POS, can contain receipts for purchase orders and transfers, as well as receipts processed with carton information. Therefore, prior to running the rcvupd module, tsfiupd module, and ctuiupd module, a Purchase Order Receipt, Transfer Receipt, and Carton Receipt Parse (tsfparse) module must be run to split out the PO receipts, the transfer receipts and the carton receipts into three files to be processed by rcvupd, tsfiupd and ctuiupd, respectively.

The tsfiupd module processes receiving detail information of transfer shipment captured by an external source. The receipt transactions provide feedback on existing transfers and shipments, update perpetual inventory for the destination location, and closes the transfers and shipments if criteria for closing is met, etc.

The ctuiupd module processes receipt transactions for carton shipments, which could have been created for either a PO or a transfer. If the carton shipment is for a transfer, this module performs transfer receiving logic for all items in the carton, as in the tsfiupd module, and updates the quantity received to be equal to quantity shipped or quantity expected for transfer detail and shipment detail records.

The Reserve Inventory for Stock Allocations (tsfresv) module is used to reserve inventory for Stock Allocation type of transfers according to its reserve date, based on the stock availability at the warehouse. If there is not enough stock to fill the needs for all the allocated location, it distributes the quantity based on the allocation method specified.

The Combine Warehouse Originated Transfers (tsfcomb) module is used to combine transfers with SR, PO and SA type into a single transfer of Combined Transfer type. Transfers from warehouses to stores which are scheduled to be shipped the following day or to other warehouses (no ship schedule is set up for warehouses) are eligible to be combined. After the transfers have been combined, 'PO' and 'SR' types are deleted, and the status of the 'SA' records are updated to 'M'.

The Approved Warehouse Transfers Download (tsfdnld) module processes all warehouse transfers that are approved, with a freight code of Normal or Expedite and have a release date equal to or less than tomorrow. If the destination location is a store, the store must be on the ship schedule to be shipped tomorrow. Shipments are created for these transfers and the shipment information is downloaded into a file to be used by an external WMS. Transfer status will be updated to 'E' (Extracted).

The Transfers Destined to a Warehouse (tsfdcdld) module processes all shipments that are in input status, with a warehouse as the final destination. The module will create a flat file that contains the shipments to download to the warehouse management system. At the end of the processing the shipment status is changed to 'E'xtracted.

The two maintenance modules: Stock Allocation Transfer Record Purge (tsfalprg) and Transfer Record Purge (tsfpgr) should be run monthly. The first purges SA type of transfers, which are closed (status = 'C'), released for shipment (status = 'M') or deleted (status = 'D'). This module will also create a file of the deleted transfers to be downloaded to the warehouse management system for maintenance purposes. The second purges all other types of transfers with status of closed, or deleted. Records will be deleted based on the number of months of transfer history to be retained, defined at the system level. This second module will also create a file of the deleted transfers to be downloaded to the warehouse management system for maintenance purposes.

Module Flow and Scheduling

The transfer batch modules tsfparse, tsfoupld, lifbolup (only if interfacing with RDM), tsfiuld, ctniupld, tsfresv, tsfcomb and tsfdnld should be run on a daily schedule as explained above. The tsfparse module must be run before tsfiupld, which is followed by ctniupld. The tsfresv module should be run before the tsfcomb module, which is followed by the tsfdnld module.

The tsfpgr and tsfalprg modules can be scheduled to run at any time, however, practical usage is to run them only once at the end of each month at the end of all batch cycles.

The following diagram 1 outlines the individual modules that are associated with the Transfers dialog:

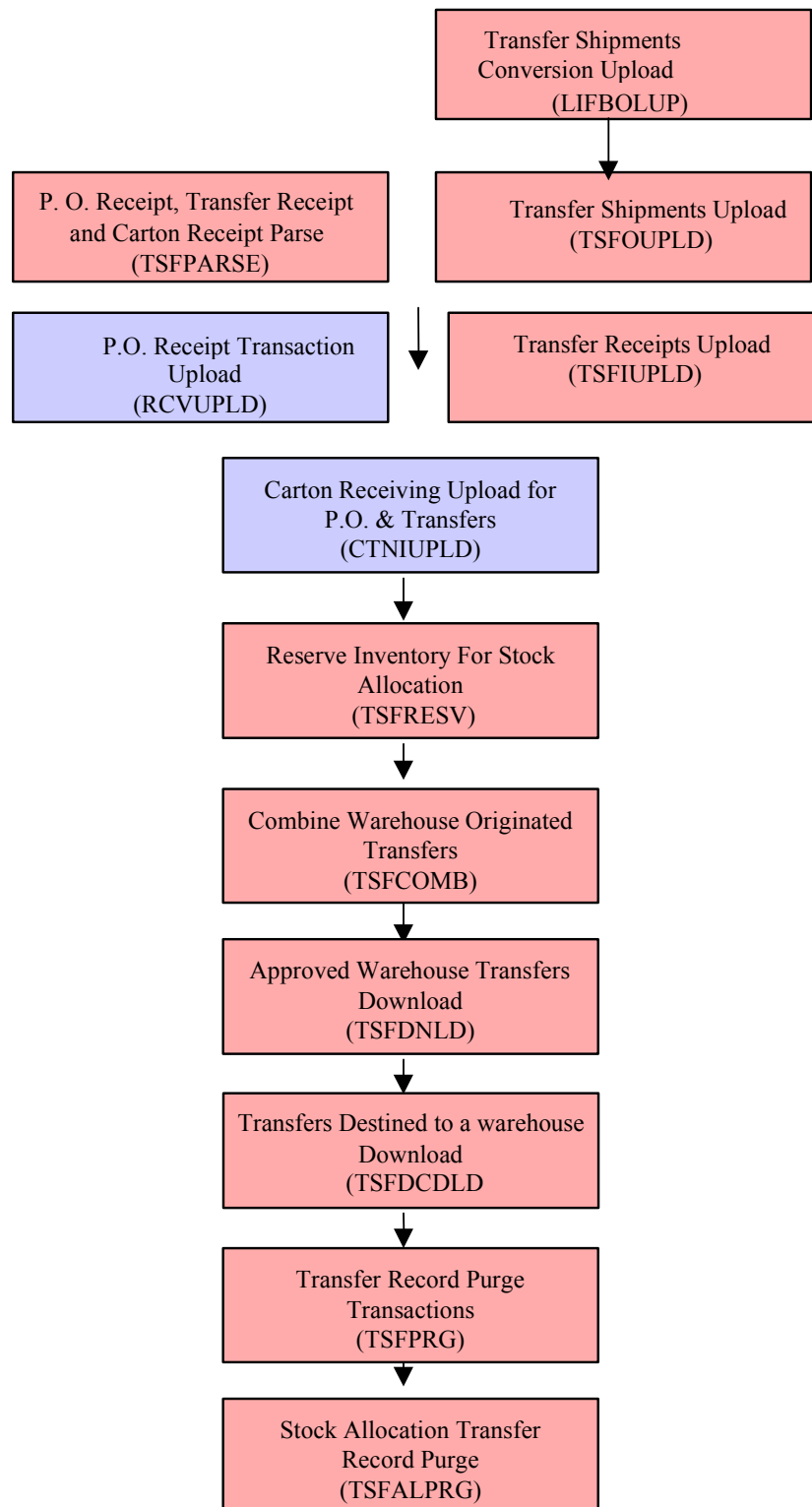


Diagram 1

Transportation

Functional Overview

The Transportation batch component is executed in order to process data from the Transportation Consolidator.

The Consolidator will send Order/Item, Vessel/Voyage/Estimated Departure Date, Container, Invoice and Bill of Lading level information which is the basis for a Pro-Forma Customs Entry. The batch component will process all of these levels and insert, update and delete information into the Transportation table.

Module Flow and Scheduling

The processing of Consolidator information (tranupld) should be run at least daily to process the files sent from an importer's consolidators.

The following diagram 1 outlines the individual modules that are associated with the Transportation dialog:

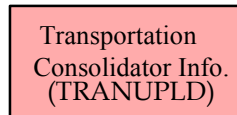


Diagram 1

VAT Maintenance

Functional Overview

The Value Added Tax (VAT) Rate Maintenance (vatdlxpl) module is executed in order to update VAT information for each item associated with a given VAT region and VAT code.

Module Flow and Scheduling

The VAT Rate Maintenance (vatdlxpl) module can be run on an ad hoc basis; however, it must be run in Phase 1 of the batch schedule before any pricing modules are executed.

The following diagram 1 outlines the module associated with the VAT Maintenance dialog:

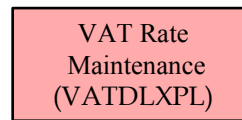


Diagram 1

Chapter 16 - Batch Master Designs

Allocation Update Download [allocupd]

Design Overview

This program will send updated pricing information to the warehouse for the items that will be allocated to stores. The allocupd.pc program will get price change information for any allocations, which have been created, and write the information to stock order and stock allocation flat files. This program will ensure that any SKU/store unit retail information that is changed after the allocation has been downloaded will be updated in Nautilus system.

The new batch program will loop through the price_hist table, selecting records whose unit retail will change tomorrow, and transaction type is in 4 or 11. Any allocations that have been created for the sku/store combination will then be downloaded with the new retail.

The allocation update download program will also produce a file that contains the ticketing and retail information for non sellable pack items that will be cross docked to store locations. This will allow the warehouse to correctly ticket the component items before the merchandise leaves to its final store destination.

Scheduling Constraints

Processing Cycle:	N/A
Scheduling Diagram:	This program should always be run after pcent.pc and after ordrev.pc
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	multi threading available

Restart/Recovery

The logical unit of work is a row from the price_hist table, selecting information for all sku/store combinations that have a price change in effect for tomorrow. This program will contain restart recovery and multi threading based on store. The driving cursor is as follows:

```
select distinct 'S', ---"normal" sku, which includes staple, fashion sku, and pack
item
```

```
    r.unit_retail,
```

```
    0 pack_no,
```

```
    r.sku,
```

```
    r.store
```

```
from price_hist r
```

```
where r.tran_type in (4, 11)
```

```
    and r.action_date = TO_DATE(:os_tomorrow, 'YYYYMMDD')
```

```
UNION ALL      /* this union selects to find if there were price changes for the
skus in a pack item */
```

```
Select distinct 'P', ---- component pack item indicator
```

```
    r.unit_retail,
```

```
    vpq.pack_no,
```

```
    vpq.sku,
```

```
    r.store
```

```
from price_hist r,
```

```
    v_packsku_qty vpq,
```

```
    packhead ph
```

```
where r.sku      = vpq.sku
```

```
    and ph.pack_no = vpq.pack_no
```

```
    and ph.sellable_ind = 'N'
```

```
    and r.tran_type in (4, 11)
```

```
and r.action_date = TO_DATE(:os_tomorrow, 'YYYYMMDD');
```

Retrieve all skus that have price changes in effect tomorrow. This information will be written to the Stock Order and Stock Allocation detail files. Also, get all of the component skus of the non sellable packs that have price changes. The component retail changes will be written to the component ticketing file.

Program Flow

N/A

Function Level Description

init()

- Declare restart variables
- Get system date.
- Open output files.
- Write FHEAD record to files.
- Call Init_buffers().

init_buffers()

- Set up format strings for outputting FHEAD, FDETL and FTAIL records.

process()

Declare a cursor to retrieve any allocations that contain that sku/store combination.

```
select ad.alloc_no,
```

```
       ah.order_no
```

```
       ad.qty_allocated
```

```
from alloc_header ah,
```

```
       alloc_detail ad
```

```
where ah.alloc_no = ad.alloc_no
```

```
and ad.store = rec.store --- from price hist
```

```
and ah.sku = rec.sku --- from price hist
```

and ad.qty_allocated > NVL(ad.qty_transferred, 0)

order by order_no;

Declare a cursor to check for order revision.

Select 'X'

From ordhead_rev

Where order_no = :os_order_no

And origin_type = 'V';

For each row meeting our criteria from the price_hist table.

- Default action_type to 'M'
- Default priority to '1'
- Default expedite_flag to 'N'

Loop through price_hist records (fetch rows into array, equal to commit max counter).

For each sku/store combination found on price_hist, get allocation details for the sku/store.

If the order number on the allocation has changed (the order number will be initialized to 0) from the previous order number, then check if a revision exists for that order. If a revision does not exist for an order, then we will not send allocation information to Logistics, because the order has not been downloaded yet.

If order revision exists,

 If the sku is not a component sku (ie. indicator = 'S')

 Fetch the ticket type from the item_ticket table where the po_print_type = 'R' (ie. ticket at receiving location). If no ticket is found, then the ticket type will default to '0000'.

Write detail record (FDETL) to the Stock Order and Stock Allocation detail files for the new unit retail created in RMS.

 If indicator = 'P' (ie. processing a pack item component)

 Write detail record (FDETL) to the Component Ticketing file for the new unit retail created in RMS.

final()

- Write file trailer (FTAIL), copy temporary file to final file, close files.

I/O Specification

All character variables are right-padded with blanks and left justified; all numerical variables are left-padded with zeroes and right-justified. Missing variables are blank.

Stock Order file

Record Name	Record	Default value	Field type	Description
File Header	Detail file identifier	FHEAD	Char(5)	Identifies the header line
	Line number	Incremented internally	Number(10)	sequential line number
	Program descriptor	STOR	Char(5)	Identifies the program
	Create date	YYYYMMDDH H24MI	Char(12)	File create date
File detail	File record descriptor	FDETL	Char(5)	Identifies the detail line
	Line number	Incremented internally	Number(10)	sequential line number
	Action_type	'M'	Char(1)	Add, modify, or delete action type
	Location	alloc_header.wh	Number(4)	From Warehouse location
	Transaction day date/time	Sysdate	Datetime(12)	system date
	distribution number	'A' + alloc_no	char(9)	Allocation number. Prefix 'A' for alloc
	Download comment	NULL	Char(30)	Comment to be printed on the label (for future use)
	Pick_not_before_date	Not_before_date	Date(8)	Not_before_date of the order
	Pick_not_after_date	Not_after_date	Date(8)	Not_after_date of the order
	Event code	Promotion or NULL	Char(6)	Promotion's event number

Record Name	Record	Default value	Field type	Description
	Event description	Prom_desc or NULL	Char(25)	Event description
	priority	1	Char(4)	Priority
	Order Type	code_detail.code	Char(9)	Type of Order: PO or PREDIST. (Taken from alloc_header.order_type)
	Break by Distro	'N'	Char(1)	Controls the mixing of orders (distros) in a container
	Carrier Code	NULL	Char(4)	Code of the carrier for the order
	Carrier Service Code	NULL	Char(6)	Carrier's service code for the delivery, First Class, and son on (Future Use)
	Route	NULL	Char(10)	Route specified for the delivery
	Ship Address Description	NULL	Char(30)	The description (such as the store name). This is the first line of the address block
	Ship Address Line 1	NULL	Char(30)	Shipping Address Line 1
	Ship Address Line 2	NULL	Char(30)	Shipping Address Line 2
	City	NULL	Char(25)	Shipping City
	State	NULL	Char(2)	Shipping State
	Zip	NULL	Char(9)	Shipping Zip
	Billing Address Description	NULL	Char(30)	The Description (such as Company name). This is the first line of the address block.
	Billing Address Line 1	NULL	Char(30)	Billing Address Line 1
	Billing Address Line 2	NULL	Char(30)	Billing Address Line 2

Record Name	Record	Default value	Field type	Description
	Billing Address Line 3	NULL	Char(30)	Billing Address Line 3
	Amount 1	NULL	Number(8,2)	Amount Charge 1
	Amount 2	NULL	Number(8,2)	Amount Charge 2
	Amount 3	NULL	Number(8,2)	Amount Charge 3
	Po number	'P' + Alloc_header.order_no	Char(9)	Unique identifier of the purchase order, prefixed with 'P'
File trailer	File record identification	TTAIL	Char(5)	File trailer identifier
	Line number	Internally incremented	Number(10)	Sequential line number of file
	Number of transaction lines	Internally determined	Number(10)	Total number of transactions (not including FHEAD and FTAIL)

Stock Allocation file

File Header:

Record	Default value	Field type	Description
File line identifier	FHEAD	Char(5)	identifies file record type
Line number	Begins at 0000000001	Number(10)	identifies file line number
Program descriptor	STAL	Char(5)	identifies the program
Create date	YYYYMMDD HH24MI format	Char(12)	file create date

File Detail:

Record	Default value	Field type	Description
Detail file identifier	FDETL	Char(5)	Identifies the Detail line
line number	Incremented internally	Number(10)	sequential line number
Action_type	'M'	Char(1)	type of record is

Record	Default value	Field type	Description
			Modify
Location	alloc_header.wh	Number(4)	From Warehouse location
Transaction day date/time	sysdate	Datetime(12)	system date
distribution number	A' + alloc_no	char(9)	Allocation number. Prefix 'A' for alloc
Item id	alloc_header.sk u	Char(16)	Unique item identifier
requested unit qty	qty_allocated	Number(12,4)	quantity allocated
destination id	price_hist.store	Number(4)	Allocation location
price	price_hist.unit_r etail	Number(7,2)	Retail price
print upc flag	'N'	char(1)	Print upc flag
ticket type	item_ticket	Number(4)	Ticket type
priority	1	Char(4)	Priority
expedite flag	'N'	char(1)	Expedite flag

File Trailer:

File Line identifier	FTAIL	Char(5)	Identifies the trailer line
line number	Incremented internally	Number(10)	sequential line number
number of transaction lines	Total number of detail lines	Number(6)	total number of detail lines in file (not including FHEAD and FTAIL)

Component Ticketing file layout:

File Header:

File Line identifier	FHEAD	Char(5)	Identifies the trailer line
Line number	0000000001	Number(10)	identifies file line number
Program descriptor	CPTT	Char(4)	identifies the program
Create date	YYYYMMDD HH24MISS	Char(14)	file create date

File Detail:

file record descriptor	FDETL	Char(5)	Detail line descriptor
line number	Incremented internally	Number(10)	sequential line number
Action_type	'A'	Char(1)	'A'dd, 'M'odify, 'D'eleate
Location	alloc_header.wh	Number(4)	location that items will be allocated from
Transaction date/time	vdate	Datetime(12)	date/time created in RMS
distribution number	alloc_header.alloc_no	char(9)	Unique identifier of the distribution.
Master item id	alloc_header.sku	Char(16)	Unique identifier of the pack item
Dest Id	alloc_detail.store	Number(4)	Identifier of the ship destination
Component Item ID	v_packsku_qty.sku	Char (16)	item identifier of the component
price	price_hist.unit_retail	Number(7,2)	Price of the merchandise.

File Trailer:

file record identification	FTAIL	Char(5)	File trailer
line number	Incremented internally	Number(10)	sequential line number
number of transaction lines	Total number of detail lines	Number(6)	total number of transaction lines in file (not including FHEAD and FTAIL)

Technical Issues

N/A

Advanced Shipment Notification Download [asndnld]

Design Overview

The WMS system requires an advanced shipment notification (ASN) file to process receipt transactions. The file is created by this module, *asndnld*. This module will be run daily, with WMS ASNs being created based on information extracted from Retek shipment/order tables.

The file layout to be used is specified below.

Scheduling Constraints

Processing Cycle:	Daily
Scheduling Diagram:	Run after ediupasn.pc and after ordrev.pc
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	Shipment (only allowed 1 thread)

Restart Recovery

The Logical Unit of Work will be a shipment.

```
EXEC SQL DECLARE c_shipment CURSOR FOR

SELECT shipment.shipment,
       shipment.order_no,
       shipment,ext_shipment,
       TO_CHAR(shipment.ship_date, 'YMMDDHH24MISS'),
       shipment.wh

FROM shipment

WHERE shipment.ship_origin = ('0', '6', '1')

AND shipment.status = 'T'

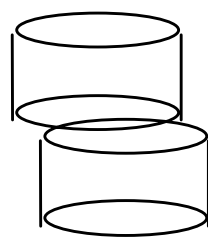
AND shipment.wh != -1

AND (shipment > NVL(:ora_restart_shipment, -999) OR
     (shipment = :ora_restart_shipment))

ORDER BY shipment;
```

Program Flow

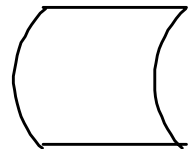
SHIPMENT, SHIPSKU



asndnld.pc

Process all shipment with
origin_code in ('0','6','1')
and status = 'T'
and wh != -1

ASN Download File



Shared Modules

PRICING_ATTRIB_SQL.GET_RETAIL(): get the unit retail from item_zone_pricing table for a sku/store.

RECEIVE_SQL.DIST_PARTIAL_SHIP(): calculates the distribution quantity

Function Level Description

init()

The init function will initialize the restart variables. It will also set up the temporary and final files as well as setup the format string for the records that need to be written to the file. Memory will be created for the structure that will hold the values fetched from the table, as well as calling the function that writes the standard header record to the temporary output file.

process()

The process function will do an array fetch of all the shipments that have an origin code of ASN Shipment ('0'), ASN UCC - 128 ('6') or a Manual Shipment ('1') that are in input status, for a warehouse location. The process function will loop through all the shipments that meet this criteria. For each shipment an Advanced Shipment Notice Header Record will be written to a temporary buffer variable.

If the shipment is a carton shipment, then (ie. carton indicator = 'Y'), set the asn_type = 'C', otherwise default the asn_type = 'P'. The action type will always be defaulted to 'A' and the carrier code and container qty will be defaulted to NULL. The ASN nbr will be the PO number joined with the shipment number retrieved from the main shipment cursor.

Loop through all of the shipsku records associated with the current shipment.

For each item in the shipment the, quantity expected (shipsku.qty_expected) and the quantity to cross-dock(min(qty_to_allocate, shipsku.qty_expected)) will be determined. If the cross-dock quantity is greater than 0, then the cross-dock flag will be set to 'Y', otherwise it will be 'N'

After the cross dock check has been made, three flat files will be written with the following guidelines:

If carton exists, then (creation of ASN container and container/item records)

Call process_carton()

If it is not a carton shipment, then (creation of ASN item file)

Call process_non_carton()

Update the shipment.status_code = 'E' extracted for the shipment.

If no errors occurred while processing the shipment details, the temporary buffer holding the Advanced Shipment Notice Header Record, and all of the Advanced Shipment Notice Detail records will be written out to the temporary file. After processing all the shipments selected in each array fetch the restart_commit and restart_file_write function will be called. The final step in this function is to call the function that writes the Standard Trailer record to the file.

Process_carton()

This function will process all of the carton shipments.

If premark_ind = 'Y', then

Retrieve the location from the carton table for the specified carton value.

If it is a customer order, then set the destination as the warehouse from the shipment table. Otherwise, set the destination as the store or warehouse of the allocation found on the carton table.

Else

Note: Non premark cross dock ASNs will be sent to Logistics through container and container/item records, however, there will be no true cross docking. This is due to the fact that only premark orders will have distribution locations for each carton sent from the supplier. Therefore, the non premark carton ASN will be received at a carton level, but must be put into “stock” at the warehouse, before it can be distributed. The receiving process will actually create the transfers for the merchandise.

Set the destination value as the warehouse (ie. shipment.location).

End If;

Call get_unit_retail() to get the unit retail of the item.

Call get_ticket_type() to get the item's ticket type.

If the carton number changes then write TCRTN record for the container by calling write_tcrtn(). The TCRTN should only be written once for every container retrieved.

Write the TCNIT record for each sku in the carton. By calling write_tcnit(). This function will write the ASN container/item record for the file. The destination id equals the value set in the above logic. The sku field will be the shipsku.sku record, the container id will be the shipsku.carton, and the unit qty will be the shipsku.qty value. There could be multiple records written for each container.

Process_non_carton()

This function will process all of the non carton shipments.

If cross dock, then

Loop through all of the allocation detail records for the order on the shipment.

If the shipsku quantity is less than the allocation quantity expected, call `receive_sql.dist_partial_ship` package. This package will calculate the quantity to distribute. If there is a split shipment (ie. supplier is sending 50 units now and 50 items later), then the quantity will be calculated based upon the allocation method retrieved from the allocation. A running total will be kept to keep track of how much has been allocated.

If the shipsku quantity is greater than the allocation quantity expected, allocate the shipment and calculate the balance of the shipment to be written after the loop ends.

If the shipsku quantity equals the allocation quantity expected, allocate the quantities of the shipment. No balance will need to be calculated.

If it is a customer order, then the destination is set to be the warehouse. Otherwise, the destination will be set based on the allocation location.

Call `get_unit_retail()`

Call `get_ticket_type()`

Call `write_titem()` function. Write TITEM record for the ASN item. A record will be written for each location found on the allocation for the ASN item. Destination id equals the allocation location (ie. store or wh), UNLESS if it is a customer order, then the destination id is the shipment warehouse, and the unit qty equals the quantity passed back from the package.

End looping through the allocation locations.

If there is a balance remaining after allocations, write the TITEM record for the balance by calling `write_titem()`, using the shipment location as the destination.

Else the order is not cross docked.

Set the destination location equal to the warehouse.

Call `write_titem()` function. Write TITEM record for the ASN item. Destination id equals the shipment.location (ie. this value should be the same as `ordloc.wh`). The unit qty will equal the `shipsku.expected_qty`.

End if;

`Write_titem()`

This function will write the TITEM records for the asn file.

`Write_tcrtn()`

This function will write the TCRTN records for the asn file.

`Write_tcnit()`

This function will write the TCNIT records for the asn file.

Get_ticket_type()

Get the ticket type for the item from the item_ticket table where the po_print_type = 'R' (print at the time of receipt). There may be several ticket types for the item with 'R' po_print_type, therefore get the first ticket type in the fetch.

Get_unit_retail()

Get Unit retail for the item/location from the item_zone_price tables by calling the pricing_attrib_sql.get_retail package.

write_std_header()

This function will format and write the Standard Header Record to the temporary output file.

write_std_trailer()

This function will format and write the Standard Trailer Record to the temporary output file.

final()

This will call the restart_file_write and restart_close library functions.

I/O Specification

Standard Header Record - One per file batch file

Field Name	Field Type	Default Value	Description
File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
File Line Sequence	Number(10)	specified by external system	Line number of the current file
File Type Definition	Char(4)	ASND	Identifies file as 'ASN header download'
File Create Date	Date	System date	date file was written by external system

ASN Transaction Header File: One record per each ASN

Field Name	Field Type	Default Value	Description
File Type Record Descriptor	Char(5)	THEAD	Identifies file record type
File Line Sequence	Number(10)	specified by external system	Line number of the current file
Transaction set control	Number(14)	specified by external system	Used to force unique transaction check
Location(DC)	Number(4)	Shipment.location	Code for the DC
Action Type	Char(1)	'A' (hardcode)	Record code
Transaction Date/Time	YYYYMMDD HHMM	System date	Date/time created on host
ASN Nbr	char(20)	shipment.order_no + shipment.shipment	Unique identifier of the ASN. By combining order number and shipment number will create a unique number to save in Nautilus. PO (8) + shipment (9) => The PO will need to be zero padded to the left to fill the full 8 bytes. The shipment will need to be padded to the left with zeros to fill the full 9 bytes. This is important for receiving upload, when

Field Name	Field Type	Default Value	Description
			we strip off the shipment number from the ASN number. 3 null pads are added to the far right (after the shipment number).
ASN Type	Char(1)	'C'ontainer level ASN, 'P'urchase Order level ASN	If shipsku has carton, then 'C' else 'P' for purchase order asn.
Carrier Code	char (4) optional	NULL	Contains the courier that will deliver the shipment
Container qty	Number(6)	NULL	Number of containers in this ASN

ASN Container File: One record per each container level ASN

Field Name	Field Type	Default Value	Description
File Type Record Descriptor	Char(5)	TCRTN	Identifies file record type. Carton record
File Line Sequence	Number(10)	specified by external system	Line number of the current file
Transaction set control	Number(14)	specified by external system	Used to force unique transaction check
Detail sequence number	Number(6)	specified by external system	Used to force unique detail check. Assigned to detail records in transaction.
Location(DC)	Number(4)	Shipment.location	Code for the DC (not sent to Logistics)

Field Name	Field Type	Default Value	Description
Action Type	char(1)	'A' (hardcode)	Record Code
Transaction Date/Time	YYYYMMDDH HMI	System date	Date/Time created on host
UCC Container ID	char(20)	Shipsku.carton	The UCC-128 carton number for shipments originating from the ASN process as carton shipments. This field will be zero for all shipments that are not at a carton level
ASN Nbr	VARCHAR(20)	Shipment.order_no + Shipment.shipment	Unique identifier of the ASN
PO Nbr	char(9)	Shipment.order_no prefixed with 'P'	Purchase Order identifier
Dest ID	number(4)	Ordloc.location OR alloc_detail.store/wh	Identifier of the shipping destination Contains the location that the shipment will be delivered to. If not cross-dock ordloc table; else if cross dock then alloc_detail table
Cube	number(6.2) 999999.99	NULL	Container Cube (dimensionless)
Weight	number(4.3) 9999.999	NULL	Container Weight (dimensionless)
Expedite Flag	char(1) Required ('Y' or 'N')	'N' (hardcode)	Flag indicating whether the order should be shipped via normal or expedited carrier service
Lot_nbr	Char(12)	NULL	Dye lot number.

ASN Container Item File: One record for each item on an ASN container

Field Name	Field Type	Default Value	Description
File Type Record Descriptor	Char(5)	TCNIT	Identifies file record type. container/item record
File Line Sequence	Number(10)	specified by external system	Line number of the current file
Transaction set control	Number(14)	specified by external system	Used to force unique transaction check
Detail sequence number	Number(6)	specified by external system	Used to force unique detail check. Assigned to detail records in transaction.
Location(DC)	Number(4)	Shipment.location	Code for the DC (not sent to Logistics)
Action Type	Char (1)	'A' (hardcode)	Record Code
Transaction Date/Time	YYYYMMDD HHMM	System date	Date/Time created on host
UCC Container ID	char(20)	Shipsku.carton	The UCC-128 carton number for shipments originating from the ASN process as carton shipments. This field will be zero for all shipments that are not at a carton level
Item ID	Char(16)	Shipsku.sku	Contains the SKU associated with the shipment
Requested Unit	Number(8)	Shipsku.qty_	The number of

Field Name	Field Type	Default Value	Description
Qty		expected	items expected to be received based on the associated order number of on the supplier's ASN for this SKU/shipment combination
Distribution Number	char(9)	Alloc_header.alloc_no (prefixed with 'A')	Unique identifier for the distribution. If no distribution exists, then NONE will be written.
Ticket Type	Number(4)	Item_ticket.ticket_type_id	Type of ticket refers to ticket type table. The ticket type with printing at the time of receipt indicator.
Unit retail	Number(7, 2)	Item_zone_price.unit_retail	Price of merchandise

ASN Item File: One record for each PO, Destination, or Item combination on an PO level ASN

Field Name	Field Type	Default Value	Description
File Type Record Descriptor	Char(5)	TITEM	Identifies file record type. Item record
File Line Sequence	Number(10)	specified by external system	Line number of the current file
Transaction set control	Number(14)	specified by external system	Used to force unique transaction check
Detail sequence number	Number(6)	specified by external system	Used to force unique detail check. Assigned to detail records in transaction.
Location(DC)	Number(4)	Shipment.location	Code for the DC (not sent to Logistics)
Action Type	Char (1)	'A' (hardcode)	Record Code
Transaction Date/Time	YYYYMMDD HHMM	System date	Date/Time created on host
ASN Nbr	char(20)	Shipment.order_no + Shipment.shipment	Contains the unique ASN number identifying a specific shipment of goods within the system.
PO Nbr	char(9)	Shipment.order_no (with 'P' prefix)	Purchase Order identifier
Item ID	Char(16)	Shipsku.sku	Unique identifier of the item
Dest ID	Number(4)	Ordloc.location OR alloc_detail.store/wh	Contains the location that the shipment will be delivered to. If not cross-dock ordloc table; else if cross dock then alloc_detail table

Field Name	Field Type	Default Value	Description
Requested Unit Qty	number(8)	Shipsku.qty_ expected	The number of items expected to be received based on the associated order number of on the supplier's ASN for this SKU/shipment combination
NEW FIELD Distribution number	Number(9)	Alloc_header.alloc_no (prefixed with 'A')	Unique identifier for the distribution
NEW FIELD Ticket type	Number(4)	Item_ticket.ticket_type_id	Type of ticket refers to ticket type table. The ticket type with printing at the time of receipt indicator.
NEW FIELD Unit retail	Number(7, 2)	Item_zone_price.unit_retail	Price of merchandise

Transaction Trailer: One record per transaction header (ie. THEAD)

File Type Descriptor	Char(5)	TTAIL	Identifies file record type
File Line Sequence	Number(10)	incremented internally	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: One record per file header (ie. FHEAD)

File Type Descriptor	Char(5)	FTAIL	Identifies file record type
File Line sequence	Number(10)	Specified by external system	Current line number.
Number of transaction lines	Number(10)	Specified by external system	Total number of lines in file, excluding FHEAD and FTAIL

Audit Purge Process [audprg]

Design Overview

The audit purge process truncates auditing tables based on the purge frequency code specified on the audit table. Valid values for purge frequency include:

- 'D' Daily
- 'W' Weekly
- 'M' Monthly
- 'H' Every six months
- 'Y' Yearly

This program accesses the following database tables.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
AUDIT_TBL	No	Yes	No	No	No
CALANDAR	No	Yes	No	No	No
PERIOD	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE AD-HOC (daily)

Scheduling Diagram: This job will be scheduled depending on the type of information to be audited in the system.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Audit Logic Information Edits [audsys]

Design Overview

This program adds or removes audit logic from a given table. A table that has an audit request raised against it will have an audit table created to hold all audit details for any inserts, updates or deletes of data. This audit table will hold the key values of the master table and the username and date of the audited transaction. Additional fields to be audited may be added or removed at the user's request. The audit module will create a database trigger to be applied to the master table. The audit table and the database trigger are automatically promoted to the database and so this program must be executed with DBA privileges.

Tables affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
AUDIT_TBL	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle:	PHASE AD-HOC (daily)
Scheduling Diagram: has been requested.	Needs to be scheduled only when new audit info
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A (single threaded)

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Cost Event Purge [ccprg]

Design Overview

The final pricing batch program is the Cost Change Purge (ccprg) module. This program is responsible for removing old cost changes from the system. Cost changes are removed from the system using the following criteria:

- the status of the cost change is Delete, Canceled, or Extracted
- the status of the price change is Rejected and the effective date of the cost change has met the requirement for the number of days that rejected cost changes are held.

Note: The number of days that rejected price changes are held is determined by the system option, number_days_rejects_held on the UNIT_OPTIONS table.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
COST_SUSP_SUP_DETAIL	No	No	No	No	Yes
COST_SUSP_SUP_HEAD	No	No	No	No	Yes
PERIOD	No	Yes	No	No	No
PRICE_EVENT_SKU	No	No	No	No	Yes
UNIT_OPTIONS	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE AD-HOC (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

```
SELECT cssh.cost_change
      FROM cost_susp_sup_head cssh
     WHERE cssh.status in ('D','C','E') OR (cssh.status = 'R'
      AND ((TO_DATE(:tomorrow) - cssh.active_date) > :days_reject_held));
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Customs Entry download [cednld]

Design Overview

This program is used to download custom entry information from the RMS database to brokers. Each night, this program will read all Custom Entry (CE) transactions that are in a Sent status for a broker id. These transactions will be written to a flat file and the status will be changed to Downloaded. One process will run and one flat file will be written per broker.

Scheduling Constraints

Processing Cycle:	2
Scheduling Diagram:	This program must run after cefinal.pc.
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	Broker

Restart Recovery

The Logical Unit of Work for the program will be a single row from the customs entry tables. Restart/Recovery will be used for init and commit.

```
SELECT LPAD(ce_id,:oi_len_ce_id,'0'),  
  
       NVL(entry_no,''),  
  
       NVL(to_char(entry_date,'YYYYMMDDHH24MISS'),''),  
  
       entry_status,  
  
       NVL(entry_type,''),  
  
       NVL(entry_port,''),  
  
       NVL(to_char(summary_date,'YYYYMMDDHH24MISS'),''),  
  
       NVL(broker_id,''),  
  
       NVL(broker_ref_id,''),  
  
       NVL(file_no,''),  
  
       importer_id,  
  
       import_country_id,  
  
       currency_code,  
  
       LPAD(exchange_rate * 10000000000,:oi_len_exchange_rate,'0'),  
  
       NVL(bond_no,''),  
  
       NVL(bond_type,''),  
  
       NVL(surety_code,''),  
  
       NVL(consignee_id,''),  
  
       live_ind,  
  
       NVL(batch_no,''),  
  
       NVL(entry_team,''),  
  
       NVL(to_char(liquidation_amt * 10000),''),  
  
       NVL(to_char(liquidation_date,'YYYYMMDDHH24MISS'),''),
```

```
NVL(to_char(reliquidation_amt * 10000),''),  
NVL(to_char(reliquidation_date,'YYYYMMDDHH24MISS'),''),  
NVL(merchandise_loc,''),  
NVL(location_code,'')  
ROWIDTOCHAR(rowid),  
';'||to_char(ce_id)  
FROM ce_head  
WHERE status = 'S'  
AND broker_id = :os_broker_id  
AND ce_id > NVL(:os_restart_ce_id, -999)  
ORDER BY ce_id;
```

Program Flow

N/A

Function Level Description

init

This function will perform standard Retek init() function logic (restart/recovery initialization, opening files, etc.). In addition, this function should select system_options.vdate, call the size_arrays() function to allocate memory for SQL fetch arrays, call the init_buffers() function to format the record strings that are written to the output file.

process

Within a loop, the driving cursor should fetch `ce_head` records into an array. For each `ce_id` that is fetched from the driving cursor, functions to retrieve records from `ce_shipment`. All records do not need to retrieve the comments field. The records for the output file that will be written will have the following hierarchy:

FHEAD

`ce_head` (THEAD)

`ce_shipment` (TSHIP)

`ce_ord_item` (TORDI)

`transportation` (TBLAW)

`transportation` (TCONT)

`ce_lic_visa` (TLICV)

`ce_charges` (TCHRG)

`missing_doc` (TMDOC)

FTAIL

After all records have been written to the output file for the CE being processed, write the rowid (retrieved from the driving cursor) to an update array. If the transaction count is equal to or greater than the `restart_max_ctr`, call the `updated_ce_head()` function to update the `ce_head` table. Also, call `restart_commit()` and `restart_file_write()`.

Process_shipments

This function will perform an array fetch to retrieve information from the `ce_shipment` table associated to the `ce_head` record being processed and call `write_file()` to write the records to the output file. This function should call the `process_order_items()` function to retrieve all order items associated with the shipment.

Process_order_items

This function will perform an array fetch to retrieve information from the `ce_ord_items` table associated to the `ce_head`, `ce_shipment` record being processed and call `write_file()` to write the records to the output file. This function should call the `process_bl_awb_id()` (only if the `ce_ord_item.bl_awb_id = 'MULTI'`), `process_container()`, `process_license_visa()`, `process_charges()`, and `process_missing_docs` functions to retrieve all detail records associated with the shipment/order/item.

Process_bl_awb_id

This function will perform an array fetch to retrieve information from the transportation table associated to the ce_ord_item record being processed and call write_file to write the records to the output file.

Process_container

This function will perform an array fetch to retrieve information from the transportation table associated to the ce_ord_item record being processed and call write_file to write the records to the output file.

Process_license_visa

This function will perform an array fetch to retrieve information from the ce_lic_visa table associated to the ce_ord_item record being processed and call write_file to write the records to the output file.

Process_charges

This function will perform an array fetch to retrieve information from the ce_charges table associated to the ce_ord_item record being processed and call write_file to write the records to the output file.

Process_missing_docs

This function will perform an array fetch to retrieve information from the missing_doc table associated to the ce_ord_item record being processed and call write_file to write the records to the output file.

Update_ce_head

This function will perform an array update of the ce_head table, changing the status from 'S'ent to 'D'ownloaded for ce_head records that have been processed. The array size counter should be initialized to zero after the post to the database.

Size_arrays

Initially size all fetch and update arrays to the size of the restart_control.restart_max_ctr (using the calloc function). If the memory cannot be allocated, raise a Fatal error.

Init_buffers

This function will format all output strings to the output file. Every time a string is initialized, it should first be set to NULL to clear it out.

Write_line

This function will write a record to the output file for the given record type passed in as a parameter.

Final

This function should perform standard Retek batch final processing. The restart_final() function should be called, the final output file should be closed and the temporary output file should be closed.

I/O Specification

Output File

The output file should be accepted as a runtime parameter at the command line.

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	Nine leading zeroes: 0000000001	ID of current line being processed by input file.
	File Type Definition	Char(4)	CEDN	Identifies file as 'Customs Entry download'
	File Create Date	Date	Create date	date file was written by external system
CE_HEAD	File Type Descriptor	Char(5)	THEAD	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	CE ID	Number(10)	ce_head.ce_id	
	Entry No	Char (15)	ce_head.entry_no	
	Entry Date	Char(14)	ce_head.entry_date	YYYYMMDDHH24 MISS format
	Entry Status	Char(6)	ce_head.entry_status	
	Entry Type	Char(6)	ce_head.entry_type	

Record Name	Field Name	Field Type	Default Value	Description
	Entry Port	Char(5)	ce_head.entry_port	
	Summary Date	Char(14)	ce_head.summary_date	YYYYMMDDHH24 MISS format
	Broker ID	Char(10)	ce_head.broker_id	
	Broker Ref. ID	Char(18)	ce_head.broker_ref_id	
	File Number	Char(18)	ce_head.file_no	
	Importer ID	Char(10)	ce_head.importer_id	
	Import Country	Char(3)	ce_head.import_country_id	
	Currency Code	Char(3)	ce_head.currency_code	
	Exchange Rate	Number(20,10)	ce_head.exchange_rate	
	Bond Number	Char(18)	ce_head.bond_no	
	Bond Type	Char(6)	ce_head.bond_type	
	Surety Code	Char(6)	ce_head.surety_code	
	Consignee ID	Char(10)	ce_head.consignee_id	
	Live Indicator	Char(1)	ce_head.live_ind	
	Batch Number	Char(20)	ce_head.batch_no	
	Entry Team	Char(3)	ce_head.entry_team	
	Liquidation Amount	Number(20,4)	ce_head.liquidation_amt	
	Liquidation Date	Char(14)	ce_head.liquidation_date	YYYYMMDDHH24 MISS format
	Reliquidation Amount	Number(20,4)	ce_head.reliquidation_amt	

Record Name	Field Name	Field Type	Default Value	Description
	Reliquidation Date	Char(14)	ce_head.reliquidation_date	YYYYMMDDHH24 MISS format
	Merchandise Loc	Char(40)	ce_head.merchandise_loc	
	Location Code	Char(4)	ce_head.location_code	
CE_SHIPMENT	File Type Descriptor	Char(5)	TSHIP	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	Vessel ID	Char(20)	ce_shipment.vessel_id	
	Voyage Flt ID	Char(10)	ce_shipment.voyage_flt_id	
	Estimated Departure Date	Char(14)	ce_shipment.estimated_depart_date	YYYYMMDDHH24 MISS format
	Vessel SCAC Code	Char(6)	ce_shipment.vessel_scac_code	
	Lading Port	Char(5)	ce_shipment.lading_port	
	Discharge Port	Char(5)	ce_shipment.discharge_port	
	Tran Mode ID	Char(6)	ce_shipment.tran_mode_id	
	Export Date	Char(14)	ce_shipment.export_date	YYYYMMDDHH24 MISS
	Import Date	Char(14)	ce_shipment.import_date	YYYYMMDDHH24 MISS
	Arrival Date	Char(14)	ce_shipment.arrival_date	YYYYMMDDHH24 MISS
	Export Country	Char(3)	ce_shipment.export_country_id	
	Shipment Number	Number(10)	ce_shipment.shipment_no	
CE_ORDITEM	File Type Descriptor	Char(5)	TORDI	Identifies file record type

Record Name	Field Name	Field Type	Default Value	Description
	File Line Identifier	Number(8)	Incremented internally	ID of current line being processed by input file.
	Order Number	Number(8)	ce_ord_item.order_no	
	Item	Number(10)	ce_ord_item.item	
	BL AWB ID	Char(15)	ce_ord_item.bl_awb_id	'MULTI' – means multiple airway bills (otherwise a single airway bill will be retrieved)
	Invoice ID	Char(30)	ce_ord_item.invoice_id	
	Invoice Date	Char(14)	ce_ord_item.invoice_date	YYYYMMDDHH24 MISS format
	Invoice Amount	Number(20,4)	ce_ord_item.invoice_amt	
	Currency Code	Char(3)	ce_ord_item.currency_code	
	Exchange Rate	Number(20,10)	ce_ord_item.exchange_rate	
	Manifest Item Quantity	Number(12,4)	ce_ord_item.manifest_item_qty	
	Manifest Item Quantity UOM	Char(4)	ce_ord_item.manifest_item_qty_uom	
	Carton Quantity	Number(12,4)	ce_ord_item.carton_qty	
	Carton Quantity UOM	Char(4)	ce_ord_item.carton_qty_uom	
	Gross Weight	Number(12,4)	ce_ord_item.gross_wt	
	Gross Weight UOM	Char(4)	ce_ord_item.gross_wt_uom	
	Net Weight	Number(12,4)	ce_ord_item.net_wt	

Record Name	Field Name	Field Type	Default Value	Description
	Net Weight UOM	Char(4)	ce_ord_item.net_wt_uom	
	Cubic	Number(12,4)	ce_ord_item.cubic	
	Cubic UOM	Char(4)	ce_ord_item.cubic_uom	
	Cleared Quantity	Number(12,4)	ce_ord_item.cleared_qty	
	Cleared Quantity UOM	Char(4)	ce_ord_item.cleared_qty_uom	
	In Transit Number	Char(15)	ce_ord_item.in_transit_no	
	In Transit Date	Char(14)	ce_ord_item.in_transit_date	YYYYMMDDHH24 MISS format
	Rush Indicator	Char(1)	ce_ord_item.rush_ind	
	Related Indicator	Char(1)	ce_ord_item.related_ind	
	Tariff Treatment	Char(10)	ce_ord_item.tariff_treatment	
	Ruling Number	Char(10)	ce_ord_item.ruling_no	
	Do Number	Char(10)	ce_ord_item.do_no	
	Do Date	Char(14)	ce_ord_item.do_date	YYYYMMDDHH24 MISS format
	Manufacture ID	Char(18)	ce_ord_item.mfg_id	
BL_AWB_ID	File Type Descriptor	Char(5)	TBLAW	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	BL AWB ID	Char(15)	Transportation.bl_awb_id	

Record Name	Field Name	Field Type	Default Value	Description
CONTAINER	File Type Descriptor	Char(5)	TCONT	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	Container ID	Char(20)	Transportation.container_id	
	Container SCAC Code	Char(6)	Transportation.container_scac_code	
CE_LIC_VISA	File Type Descriptor	Char(5)	TLICV	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	License/Visa Type	Char(6)	ce_lic_visa.license_visa_type	
	License/Visa ID	Char(30)	ce_lic_visa.license_visa_id	
	License/Visa Quantity	Number(12,4)	ce_lic_visa.license_visa_qty	
	License/Visa Quantity UOM	Char(4)	ce_lic_visa.license_visa_qty_uom	
	Quota Category	Number(3)	ce_lic_visa.quota_category	
	Net Weight	Number(12,4)	ce_lic_visa.net_weight	
	Net Weight UOM	Char(4)	ce_lic_visa.net_weight_uom	
	Holder ID	Char(18)	ce_lic_visa.holder_id	
CE_CHARGES	File Type Descriptor	Char(5)	TCHRG	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.

Record Name	Field Name	Field Type	Default Value	Description
	Sequence Number	Number(6)	ce_charges.seq_no	
	Pack Item	Number(8)	ce_charges.pack_item	
	HTS	Char(10)	ce_charges.hts	
	Effect From Date	Char(14)	ce_charges.effect_from	YYYYMMDDHH24 MISS format
	Effect To Date	Char(14)	ce_charges.effect_to	YYYYMMDDHH24 MISS format
	Component ID	Char(10)	ce_charges.comp_id	
	Component Rate	Number(20,4)	ce_charges.comp_rate	
	Per Count UOM	Char(3)	ce_charges.per_count_uom	
	Component Value	Number(20,4)	ce_charges.comp_value	
MISSING_DOC	File Type Descriptor	Char(5)	TMDOC	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	Doc_id	Number(6)	Missing_doc.doc_id	
	Received_date	Date	Missing_doc.received_date	
File Trailer	File Type Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	File Record Counter	Number(10)	Determined Internally	Number of records/transactions processed in current file (only records between head & tail)

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

N/A

Competitive Pricing Purge [cmpprg.pc]

Design Overview

This new batch program will perform the desired deletions from the competitive price history table and the competitive shopping list table based on system variables. This program will also delete completed records from the competitive shopping list table each day.

On the system options table, the comp_price_months field will determine how many months competitive price history should be maintained before deletion. The comp_list_days field will determine how long a requested shopping list should remain on the shopping list table if it is not complete by the requested shop date.

Scheduling Constraints

Processing Cycle:	Ad Hoc (Daily)
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

N/A

Program Flow

N/A

Function Level Description

This program will mirror other purge programs such as hstprg.pc

main:

This is the main routine of the program. The standard Retek main function can be used here.

init:

Retrieve the vdate from the period table along with the comp_price_months and comp_list_days variables from the system options table.

process:

Based in the comp_hist_months variable, delete from the comp_price_hist table where the rec_date is less than the v_date less the comp_hist_months variable.

Based in the comp_list_days variable, delete from the comp_shop_list table where the shop_date is less than the v_date less the comp_list_days variable. Also delete any records from the comp_shop_list table that have been competed (i.e. the comp_retail field is not NULL).

final:

Standard final processing.

I/O Specification

No files.

Testing Scenarios

- Run program and make sure the correct fields are being deleted from the comp_price_hist table and the comp_shop_list table.

Contract Maintenance and Purging [cntrmain]

Design Overview

All contracts with statuses not equal to 'Reviewed' or 'Approved' and their associated records will be deleted after a specified number of months has elapsed since their last change in status if there are no existing orders against it.

In the status maintenance function, 'Approved' closed (types A and B) contracts will be set to 'Complete' automatically. Those that are of type C and D will be set to 'Review' status automatically.

Contracts can also be set to 'Complete' manually in the contract header form. Validation on-line will only check outstanding approved orders and no orders exist in the orders tables. In 'Review' status contracts are still active and it is valid to raise new orders against them (manually or automatically). Open contracts (types C and D) will not automatically be set to 'Complete' or 'Cancelled' as this will be a manual action that takes place as part of the review process. The user will only be able to re-approve a contract in review status using the on-line functionality.

This program accesses the following database tables:

TABLE	SELECT	INSERT	UPDATE	DELETE
UNIT_OPTIONS	Yes	No	No	No
PERIOD	Yes	No	No	No
CONTRACT_HEADER	Yes	No	Yes	Yes
CONTRACT_SKU	No	No	No	Yes
CONTRACT_PROD_PLAN	No	No	No	Yes
ORDHEAD	Yes	No	No	No

Scheduling Constraints

Processing Cycle:	This module will run daily.
Scheduling Diagram: replenishment cycle.	This module needs to be scheduled before the
Pre-Processing:	None.
Post-Processing:	None.
Threading Scheme:	None.

Restart Recovery

Contracts pending deletion are read into an array, and all records in contract_cost, contract_detail, and contract_header relating to those contracts are deleted. Transactions are committed after an entire array has been read and deleted. Arrays in this program may need to be smaller than in most other array based restart/recovery programs to control the size of the rollback segment. This is because for each entry in the array records need to be deleted from three tables.

Program Flow

init

 retrieve program variables

process

 call delete_contracts function

 call reset_inactive function

delete_contracts function

 driving cursor LOOP through contracts valid for deletion on contract_header

 if contract status is C (complete) then

 validate no valid orders are attached to contract (do not delete contract if an order exists)

 delete from contract tables

 END LOOP

reset_inactive function

 update contract_header

 where contract type is A or B, approved, and passed the end-date then

 update contract_header

 where contract type is C or D, approved, and passed the end-date then

Shared Modules

N/A

Function Level Description

include std_err header file and use error handling routines in oracle.pc library.

init()

retrieve system date.

retrieve order_history_months, contract_inactive_months and contract_review_days from unit_options.

process():

call delete_contracts function

call reset_inactive function

delete_contracts():

loop

driving cursor on contract_header table with the following where clause: status in ('W', 'S', 'C', 'X', 'D') AND months between status_date and today >= order_history_months AND orders do not exist

exit loop if no rows found.

delete from the following tables in the specified order using the contract_no selected from the cursor:

- contract_cost
- contract_detail
- contract_header

commit transaction

end loop

reset_inactive():

Update contracts where: contract_type is A or B AND approved AND today's date >= end-date then set contract_header.status = 'C', and contract_header.status_date and contract_header.completed_date to today's date.

Update contracts where: contract_type is C or D AND approved AND today's date >= end-date then set contract_header.status = 'R', contract_header.status_date and contract_header.review_date to today's date, and contract_header.review_id to 'AUTOMATIC'.

exception handling: when SQL_ERROR_FOUND, WRITE_ERROR using SQLCODE and return (-1).

end execution block.

I/O Specification

N/A

Contract Replenishment – Type ‘B’ Contracts [cntrordb]

Design Overview

Batch module creates automatically replenished orders for type B contracts

Orders will be created for all type B contract items that are ready to have orders raised against them. Contract, SKU, and location information will be selected from the contracting tables where production dates are ready to be met.

An order will be written for each contract and all of the items & locations on the contract.

Scheduling Constraints

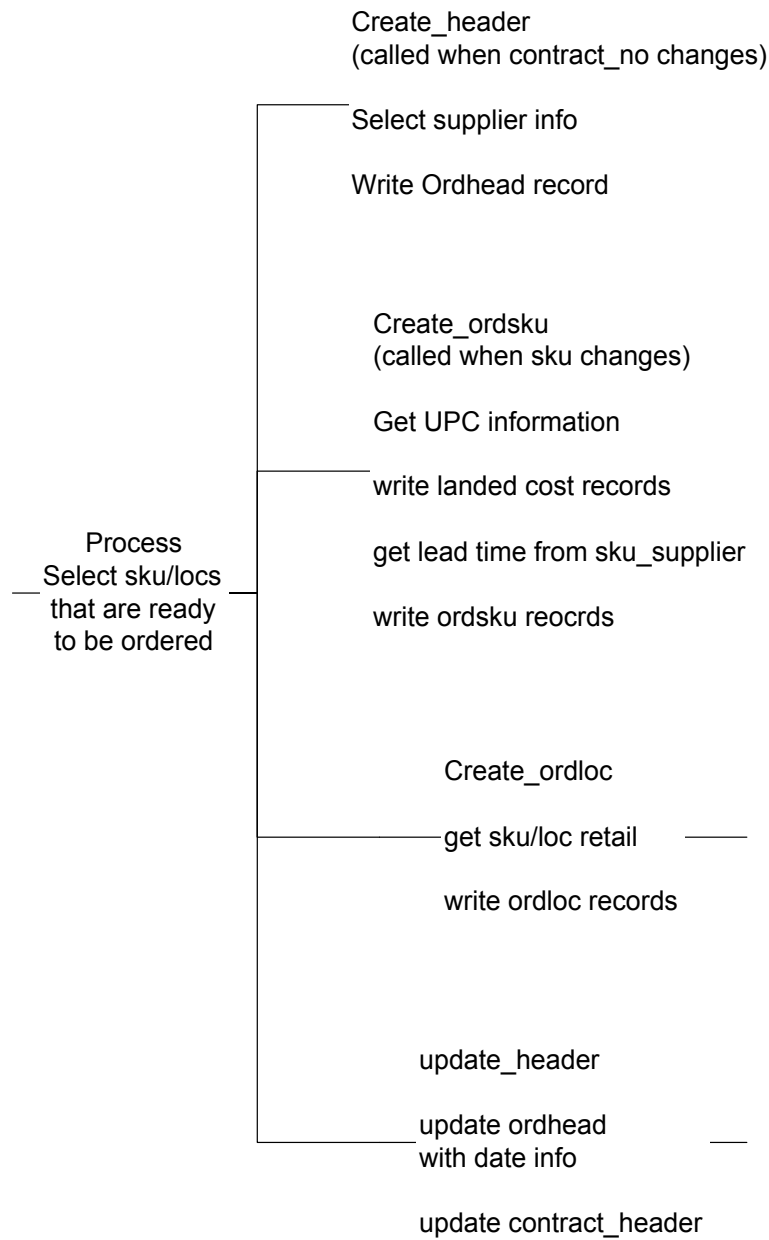
Processing Cycle:	Phase 3. Must run after rpladjs & rpladjf
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	Update of system_variables, set last_cont_order_date = vdate
Threading Scheme:	Contract_no (questionable whether this is necessary)

Restart Recovery

The logical unit of work is a unique contract. A commit will take place after the number of contracts processed is equal to the max counter from the restart_control table.

Multi-threading issue: If the contract_header table is sizable, the threading mechanism may have to be re-thought or omitted.

Program Flow



Shared Modules

N/A

Function Level Description

Init()

System level variables are selected here including the internal RMS date, the date of the last type B contract ordering run, the minimum number of days before a contracted ready date before an order can be raised, and landed cost information. The restart/recovery process should be initialized.

Process()

Contracted SKU/locations that are to have orders raised against them are selected from the contracting tables. These are only for contracts of type B, that are within the production plan time frame. An order will be created for each contract, and the create_header function will perform the inserts into the RMS ordhead table. After the order has completed, the order header and contract header information is updated by calling the update_header function.

The add_ordsku function is called to insert SKU level information into the RMS ordsku table. If the ELC indicator is “Y”, then each sku on the order should be sent to the add_cost_comp function.

The add_ordloc function is called for every record to insert location quantity and retail information into the RMS ordloc table.

create_ordhead()

Insert the order header information into the ordhead table. The NEXT_ORDER_NUMBER stored function is called to retrieve the next available order number for the insert. The package CURRENCY_SQL.GET_RATE is also called to get the exchange rate to use in the insert. The insert is in the form of an insert/select. It joins with the sup_import_attr and addr tables to get the information necessary to create the order.

Add_ordsku()

This function adds SKU level order records. If the contract does not include UPC information, the supplier’s primary UPC value is retrieved for inserting into the ordering table. The sku information, including UPC, is inserted into the ordsku table. The get_dates function is called to retrieve the lead time for the sku and supplier. These dates will be used to determine the not before and not after dates on ordhead. This functions call the sku_defaults function.

Sku_defaults()

This function call the stored procedure ORDER_SETUP_SQL.DEFAULT_ORDSKU_DOCS. This package call defaults all the required documents that are needed when creating ordsku_records.

Update_ordsku()

This function is called whenever processing for a given sku, on a given contract is completed. It updates the latest_ship_date with the correct value given the latest ready_date for the sku on the contract. The sku could have many different ready_dates and the latest ship date should reflect the latest one. The stored procedure CONTRACT_SQL.GET_UNIT_COST is call to get the unit cost to use in the insert.

Add_ordloc()

This function first checks if an ordloc record already has been inserted. If a record, exists it updates the ordered qty. If a record does not exist this function adds location level order records. The sku/location retail value is retrieved by the get_retail function. The sku/location information will be inserted into ordloc. The contracting item-level tables will be updated to reflect that the ordered quantity was ordered against that contract, sku, and production plan date.

Get_retail()

The retail value should be selected from the appropriate sku/location (item store or warehouse) table.

Get_dates()

This function determines the most earliest and latest ready dates. These dates are then used when updating ordhead and ordsku.

Add_cost_comp

This function will update the landed cost tables with the new order information. The stored package ORDER_EXPENSE_SQL.INSERT_COST_COMP is called to update the system tables. The package ORDER HTS_SQL.DEFAULT_CALC HTS is called to default the hts information for sku.

ELC_CALC_SQL.CALC_COMP is called to recalculate expenses.

Update_header()

The information (on ordhead) is updated with the derived not before and not after dates (determined by the SKU lead times).

Update_ordsku)

The latest ship_date is updated based on the greatest ready_date determined by get_dates().

I/O Specification

N/A

Technical Issues

N/A

Contract A/C/D Replenishment Processing [cntrprss]

Design Overview

This module processes staple stock requirements on the ord_temp table by sourcing the stock from the best contract or supplier. And update repl_results table to hold information about the qty of the item that is satisfied by the contracts. This module only processes the records on ord_temp which due_ind = 'Y'. This module will only be run if contracting is turned on in the system. Only system controlled replenishment stock will be integrated with contracting.

Contract type B are not processed in this module. Automatic orders for this contract type are generated in the CNTRORDB module.

Current replenishment process overview:

The requirements extraction process (rplexts) generates requirements for primary suppliers on review. Records are written to the ord_temp and ordsugg tables. The ord_temp table holds requirements detailing the supplier, department, sku, quantity required, location, and replenishment stock type (Direct to Store, Cross-dock, Warehouse stocked). The order build process (rplblds) takes these requirements and generates orders and allocations.

Contracting integration conceptual overview:

In order to integrate the replenishment process with contracting CNTRPRSS will act as an interpreter between the requirements extraction and order build modules. CNTRPRSS will take requirements (from ord_temp) and source them from the best contract/supplier in the same review cycle. If the contract cannot satisfy the whole requirement, then another contract is used to satisfy the remaining requirement. Any unsatisfied requirement is written to the ord_misseds table.

TABLES/FILES ACCESSED	SELECT	INSERT	UPDATE	DELETE
ORD_TEMP	Yes	Yes	Yes	Yes
REPL_RESULTS	Yes	No	Yes	No
PERIOD	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No
CONTRACT_SKU	Yes	No	No	No
CONTRACT_HEADER	Yes	No	No	No
CONTRACT_PROD_PLAN	Yes	No	No	No
SKU_SUPPLIER	Yes	No	No	No
SUPS	Yes	No	No	No
DUTY_RATES	Yes	No	No	No
ORD_MISSEDS	No	Yes	No	No
HDR_COMMIT_S	Yes	No	Yes	No

Background

contract_replenish_ind: The system level contract_replenish_ind indicates whether a requirement can be satisfied from outside of a contract or not (i.e., can orders be raised with non-contracted suppliers in the replenishment cycle). This flag is used in CNTRPRSS to determine when requirements should be written to ord_misseds as unsatisfied if no contract is available for sourcing.

replenishment cycles: Review cycles will be allowed at the vendor level for items replenished from the vendor, and at the store level for items replenished from the warehouse (Warehouse Stocked (W)). The user replenishes items daily, multiple times per week or in a weekly increment (every 2 weeks, every 3 weeks, etc.). By having this option at the vendor level, the user may ensure all items for a given vendor are reviewed at the same time in order to help meet vendor minimums, brackets, etc.. Also, the user may specify the week day for review (i.e. Monday, Tuesday, Wednesday...) so that he/she may better balance their workload across the week.

Note also that only primary suppliers on review are processed in the requirements extraction module (rplexts).

In this module only those suppliers on review will be considered valid for satisfying requirements (i.e., any contracts that do not have the supplier on review will not be considered). It is important, therefore, when setting-up contracts on-line that the supplier be

- the primary supplier
- on a review cycle
- active

rplexts: This program is run on a cyclic basis (in conjunction with the programs **rplblds**, **reqexts** and the report **ordsugg**) in order to maintain optimum stock levels of staple stock items. For further detail see **rplexts** design.

rplblds: To build orders during the automatic re-order cycle by consolidating the required items as generated in **rplexts**. For further detail see **rplblds** design.

Replenishment stock categories: There are three stock categories which determine how an item is sourced; Direct to Store (D) where the item flows from the vendor to the store, Cross-docked (C) where the item flows from the vendor to the store, flowing through a distribution centre, or Warehouse Stocked (W) where the item flows from the vendor to a warehouse where it is stocked and sourced at a later date by the stores. Replenishment order control determines whether or not the replenishment will be buyer controlled or system controlled. Buyer controlled versus system controlled simply determines whether a PO will be generated automatically or a report listing recommended order quantities can be run from which the user can manually create a PO. This information is maintained at an SKU location level.

Following are the various codes attached to the requirements generated in the requirements extraction module:

	wh_or_store_c	wh_or_store	pack size	store
Direct to Store (S)	1	store number	supp_pack_size	-1
Warehouse Stocked (W)	0	warehouse number	supp_pack_size	0
Cross-docked (C) -wh order requirement	0	warehouse number	supp_pack_size	-1
Cross-docked (C) - store allocation requirement	0	warehouse number	inner_pack_size	store number

Detailed Overview

The process will attempt to source the item requirement from the best contract available. The existing ord_temp supplier is then replaced/updated with the contract/supplier details.

The best contract is selected according to the following set of rules (in the rankentr procedure);

1. earliest - today + lead time
2. cheapest - unit cost (include duty for foreign suppliers if applicable)
3. closed contracts are preferred over open contracts (i.e., A over C & D)
4. contract type C is preferred over contract type D

The selected contracts are validated for the following:

Type	Production plan location	Supplier availability	Total outstanding commitment	Sku level commitment	Supplier active & on review
A		X		X	X
C			X		X
D		X	X		X

Any requirement that could not be satisfied by a contract is sourced from the best remaining supplier, excluding those with contracts. If sourcing is not permitted outside of contracts then any unsatisfied requirement is written to ord_misseds and not sourced from the remaining suppliers. The supplier is chosen on the basis of earliest delivery date and cost. If sourcing is not permitted outside of contracts or no suppliers were available then the unsatisfied commitment is written to the ord_misseds table. The report uns_reqs can be run against this table to report on the unsatisfied requirement.

Scheduling Constraints

Processing Cycle:

This module will run on the same schedule as the replenishment cycle.

Scheduling Diagram:

This module needs to be scheduled after rplexts and before rplblds.

Threading Scheme:

The threads will be split by department.

Restart Recovery

The logical unit of work will comprise the processing of a staple sku requirement on ord_temp (i.e., it is either sourced or written to ord_misseds).

Carton Receiving Upload (for P.O. and Transfers) [ctniupld]

Design Overview

The purpose of this batch module is to accept receipt information when carton number is used in the receiving. This program first calls a package function `carton_sql.receive` which determines whether the receipt transaction is for a PO or a transfer. If it is for a PO, the package function `receive_sql.recv_carton` is called, else `transfer_in_sql.carton` is called to perform the receiving functions.

Scheduling Constraints

Processing Cycle: Must be run after `tsfparse.pc` and `revupld.pc`. In a DC flow through type of operation, Transfer Out. Upload (`tsfoupld`) module should follow `ctniupld` module in order that the transfers created in `ctniupld` module be shipped out within the same day.

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Only one thread since this will process an input file.

Restart Recovery

The logical unit of work for the carton receiving upload module is the discrete carton number. If any portion of the processing for the complete carton receiving transaction fails, the entire carton must be reprocessed.

A savepoint will be issued prior to processing a new carton. If any record within the transaction fails, the whole transaction will be rolled back to the most recent savepoint. This way, the successfully processed transactions will remain posted to the database but not yet committed.

To prevent excessive rollback space usage, intermittent commits will be issued based on a commit counter. The recommended commit counter setting is 1000 records (subject to change based on experimentation). The commit counter is based on actual records processed not overall transactions, nor the number of writes to the database, since the database interactions will be a constant multiplier of the commit counter. A carton receiving transaction cannot be committed to the database until it is complete so the commit counter is viewed as a minimum threshold, that once reached, will force a commit after the completion of the current carton receiving transaction.

Error handling will be based on the logical unit of work also. If a given record within a carton receiving transaction fails, that error will be posted to the standard error log for the batch module. If the error is of a non-fatal type, all subsequent detail records within the carton receiving transaction will continue to be processed and any errors noted will continue to be posted. After processing all errors for the transaction, the entire carton receiving transaction will be rejected to a runtime specified rejection file. If a fatal error is encountered, the file pointer at the time of the last commit will have been posted to the bookmark and all transactions from the last commit will be rolled back. Processing will commence at the saved file position.

Program Flow

N/A

Shared Modules

CARTON_SQL.RECEIVE - used to determine if the carton/location that is being received is for a purchase order (PO) or for a transfer based on if the carton/location combination exists on the carton table or not. If it does, this is a transfer receipt (carton transfer receipt is a receipt against a transfer created from a cross-dock PO and the location number stored on the carton table is the final destination of the carton, i.e. the transfer receiving location), else this is a PO receipt. It then calls the appropriate function RECEIVE_SQL.RECV_CARTON or TRANSFER_IN_SQL.CARTON to process the receiving.

RECEIVE_SQL.RECV_CARTON – performs receiving for all items in a PO shipment with all of its shipsku records tied to the input carton number. For each item received, it calls the RECEIVE_SQL.ITEM package function to perform the PO receiving functions. If a ‘premark’ cross-dock PO is received, transfer record will be generated to transfer the item to the PO’s final destination.

TRANSFER_IN_SQL.CARTON - performs receiving for all items in a ‘premark’ cross-dock transfer shipment with all of its shipsku records tied to the input carton number. For each item received, it calls the TRANSFER_IN_SQL.EXECUTE package function to perform the transfer receiving functions

Function Level Description

init()

- initialize restart recovery variables
- open input file
- set up reject file

process()

- read the first record - FHEAD and validate
- in a while loop read the next record
- if the record type is FTAIL break;
- else parse the values into variables
- validate that carton number/receiving location combination exist on shipment/shipsku tables.
- call the package function CARTON_SQL.RECEIVE - include proper error handling
- if the package errors out, write record to reject file
- include a call to restart_commit
- end while
- process TTAIL and validate

final()

- write tail record to the reject file
- restart_close
- close input and reject files

I/O Specification

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	CRTN	Identifies file as 'Carton Receiving'
	File Create Date	Date	create date	date file was written by external system
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Carton Number	Char(20)	specified by external system	Specifies the unique UCC-128 Code for a carton
	Location	Number(4)		Xdock location of carton
	Transaction Date	Date	specified by external system	date the carton was received by the external system
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Total number lines	Number(10)	Specified by external system	Total number lines in file

Reject File

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will need to be created by the carton receiving module and a reject line counter will be required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and trailer record and the file line count will be equal to 0.

The reject filename should also be specified as a runtime parameter.

Error File

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

N/A

[dealcalc.pc]

Design Overview

This new batch program will calculate the net cost, net net cost, and dead net net cost for all items that are on the deal_sku_temp table (which should contain all items or items in hierarchies on deals that are on the deal_queue table, which will contain deals that are about to be approved, unapproved, or closed—any action that would potentially change which deals affect an item). All active deals for each item will be used in the calculation. Once calculated, the costs will be inserted into the deal_sku_cost table.

Scheduling Constraints

Processing Cycle:	Phase II (daily)
Scheduling Diagram:	Must be run after ditinsrt.pc, which populates the deal_sku_temp table
Post-Processing:	Call prepost to delete all records from deal_sku_temp.
Threading Scheme:	SUPPLIER

Restart Recovery

Logical unit of work: sku/supplier/origin country/start date

The driving cursor will be dynamically created depending on ordering requirements, which will be determined by deal_type_priority and deal_age_priority of system_options.

```
SELECT dst.sku,
       dst.supplier,
       dst.origin_country_id, /* DST country not DI country—if no country
given, DO expand out */
       TO_CHAR(dst.start_date,'YYYYMMDD'),
       sups.currency_code,
       isc.unit_cost,
       dh.deal_id,
       dd.deal_detail_id,
```

```

        dh.currency_code,

NVL(dst.location, -1) /* DST loc not DI loc—expand out location unless loc-
independent */

NVL(dst.loc_type,'N')

DECODE(dd.cost_appl_ind,'N',1,'NN',2,'DNN',3) cost_appl_num,

dd.deal_class,

dd.threshold_value_type,

NVL(dd.qty_thresh_buy_item, -9999),

NVL(dd.qty_thresh_buy_qty, 0),

NVL(dd.qty_thresh_recur_ind,'N'),

NVL(dd.qty_thresh_buy_target, 0),

NVL(dd.qty_thresh_get_item, -9999),

NVL(dd.qty_thresh_get_qty, 0),

NVL(dd.qty_thresh_free_item_unit_cost, 0),

NVL(dd.qty_thresh_get_type, 'Z'),

NVL(dd.qty_thresh_get_value, 0),

        TO_NUMBER(di.merch_level, 0),

        TO_NUMBER(NVL(di.org_level, 99)

FROM deal_sku_temp dst,

deal_head dh,

deal_detail dd,

deal_itemloc di,

supr,

item_sup_country isc,

v_restart_supplier vrs

WHERE      dd.deal_id = dh.deal_id

AND di.deal_id = dd.deal_id

```

```
AND di.deal_detail_id = dd.deal_detail_id

AND dh.status = 'A'

AND dh.type in ('A','P') /* only use promotional/annual, not PO specific or
vendor funded */

AND di.excl_ind = 'N'

AND sups.supplier = dst.supplier

AND isc.item = dst.sku

AND isc.supplier = dst.supplier

AND isc.origin_country_id = dst.origin_country_id

AND ((dh.close_date is NOT NULL

      AND dst.start_date BETWEEN DECODE(rebate_ind, 'Y',
      NVL(dd.rebate_active_date, dh.active_date), dh.active_date)

      AND dh.close_date)

OR (dh.close_date is NULL

      AND dst.start_date >= DECODE(rebate_ind, 'Y',
      NVL(dd.rebate_active_date, dh.active_date), dh.active_date)))

AND ((dh.supplier is NOT NULL AND dst.supplier = dh.supplier) /*
supplier hierarchy match */

      OR(dh.partner_type = 'S1' AND isc.supp_hier_lvl_1 = dh.partner_id)

      OR(dh.partner_type = 'S2' AND isc.supp_hier_lvl_2 = dh.partner_id)

      OR(dh.partner_type = 'S3' AND isc.supp_hier_lvl_3 = dh.partner_id) )

AND ( (di.merch_level = 1)

      OR (di.merch_level = 2 AND di.division = dst.division

      OR (di.merch_level = 3 AND di.group_no = dst.group_no)

      OR (di.merch_level = 4 AND di.dept = dst.dept )

      OR (di.merch_level = 5 AND (di.dept = dst.dept AND di.class =
dst.class))

      OR (di.merch_level = 6 AND (di.dept = dst.dept AND di.class = dst.class
AND di.subclass = dst.subclass))
```

```

        OR (di.merch_level = 7 AND di.style = dst.style)           --style/color
hierarchy

        OR (di.merch_level = 8 AND (di.style = dst.style AND di.color =
dst.color)

        OR (di.merch_level = 9 AND (di.style = dst.style AND ((di.size1 =
dst.size1 OR di.size1 is NULL)                               AND (di.size2 = dst.size2 OR
di.size2 is NULL)))

        OR (di.merch_level = 10 AND di.sku = dst.sku))

AND (di.org_level is NULL AND dst.chain is NULL

        AND dst.area is NULL AND dst.region is NULL

        AND dst.district is NULL AND dst.location is NULL

OR (di.org_level = 1 AND di.chain = dst.chain)

OR (di.org_level = 2 AND di.area = dst.area)

OR (di.org_level = 3 AND di.region = dst.region)

OR (di.org_level = 4 AND di.district = dst.district)

OR (di.org_level = 5 AND di.location = dst.location))

AND (di.country_id = dst.country_id OR di.country_id is NULL)

/* exclude clause here --don't fetch excluded skus */

AND (NOT EXISTS

        SELECT 'x'

        FROM deal_itemloc di1

        WHERE di1.deal_id = di.deal_id

        AND di1.deal_detail_id = di.deal_detail_id

        AND di1.excl_ind = 'Y'

AND ( (di1.merch_level = 1)

        OR (di1.merch_level = 2 AND di1.division = dst.division

        OR (di1.merch_level = 3 AND di1.group_no = dst.group_no)

        OR (di1.merch_level = 4 AND di1.dept = dst.dept )

```

```

        OR (di1.merch_level = 5 AND (di1.dept = dst.dept AND
di1.class = dst.class))

OR (di1.merch_level = 6 AND (di1.dept = dst.dept AND di1.class = dst.class
        AND di1.subclass = dst.subclass))

        OR (di1.merch_level = 7 AND di1.style = dst.style)           --
style/color hierarchy

        OR (di1.merch_level = 8 AND (di1.style = dst.style AND di1.color =
dst.color)

        OR (di1.merch_level = 9 AND (di1.style = dst.style
        AND ((di1.size1 = dst.size1 OR di1.size1 is
NULL)
        AND (di1.size2 = dst.size2 OR di1.size2 is
NULL))))

        OR (di1.merch_level = 10 AND di1.sku = dst.sku))

AND (di1.org_level is NULL AND di1.chain is NULL
        AND di1.area is NULL AND di1.region is NULL
        AND di1.district is NULL AND di1.location is NULL

OR (di1.org_level = 1 AND di1.chain = dst.chain)

OR (di1.org_level = 2 AND di1.area = dst.area)

OR (di1.org_level = 3 AND di1.region = dst.region)

OR (di1.org_level = 4 AND di1.district = dst.district)

OR (di1.org_level = 5 AND di1.location = dst.location))

AND (di1.origin_country_id = dst.origin_country_id OR di1.origin_country_id
is NULL))

AND (dst.sku > NVL(:ps_restart_sku, -999) OR /* restart on
item/supplier/country/start_date */

(dst.sku = :ps_restart_sku AND

(dst.supplier > :ps_restart_supplier OR

(dst.supplier = :ps_restart_supplier AND

(dst.origin_country_id > :ps_restart_country OR

```

```
(dst.orign_country_id = :ps_restart_country AND  
dst.start_date > :ps_restart_date))))))  
AND vrs.num_threads = :pi_num_threads  
AND vrs.thread_val = :pi_thread_val  
AND vrs.driver_value = dst.supplier  
ORDER BY dst.sku,  
dst.supplier,  
dst.origin_country_id,  
dst.start_date,  
loc,  
cost_appl_num,  
dh.type,  
dh.create_date,  
dd.application_order
```

The ORDER BY dh.type's and dh.create_date's asc/desc following rules:

- create date asc, annual before promotional (dh.type asc)
- create date desc, annual before promotional
- create date asc, promotional before annual (dh.type desc)
- create date desc, promotional before annual

Program Flow

Tables used:

Table	Select	Insert	Update	Delete
period	X			
system_options	X			
deal_sku_temp	X			X
deal_head	X			
deal_detail	X			
deal_itemloc	X			
deal_threshold	X			
deal_sku_cost		X		
item_supp_country	X			
sups	X			

Shared Modules

CURRENCY_SQL.CONVERT –convert an amount in deal currency to the equivalent amount in supplier currency if necessary, or vice versa

Function Level Description

init:

- Retrieve the vdate from the period table (use as calculation date for inserts into deal_cost table).
- Get priority indicators (deal_type_priority, deal_age_priority—these determine annual first vs. promotional first, and oldest first vs. newest first ordering for the driving cursor) from system_options.
- Allocate memory for the deal fetch and cost arrays (call size_arrays) and initialize the linked list for deal target values.
- Restart/recovery initialization.

process:

- Call `prepare_driving_cursor` to create driving cursor statement based on the system options.
- Use the driving cursor to get all active deals for each item/supplier/origin country/start date on the `deal_sku_temp` table (use an array fetch).
- For each deal/deal detail, call `get_target_threshold_value` to find the threshold value to be used in cost calculations.
- Call `calculate_cost_driver` to get the net, net net, and dead net net cost (initially for location-independent deals and then for the location-specific deals, starting from the costs already calculated for location-independent deals), and create an insert array that includes the net/net net/dead net net cost information AND the location information.
- If commit point reached, call `post_insert_delete_records` to insert the costs into the `deal_sku_cost` table FOR EACH LOCATION of the same LUW (including a record with no location if there are location-independent deals), and to delete processed records from the `deal_sku_temp` table.
- After each set of deals has been processed, call the restart commit logic.

`prepare_driving_cursor`:

Create driving cursor statement based on the system options `deal_type_priority` and `deal_age_priority`, which only affect the ORDER BY clause.

`calculate_cost_driver`:

This function will drive the process of calculating the net, net net, and dead net net cost, given information on all the deals that apply to a particular sku/supplier/origin country/start date (pass in array structs which include the target threshold value). Each deal/deal detail record is passed on to the `calculate_costs` function to do the actual calculation for each LUW + loc, that is, sku/supplier/origin country/start date/loc.

For each set of deals for a unique item/supplier/country id/start date, the desired end result is to have one record on deal_sku_cost with no location that will hold the item's costs with all location-independent deals accounted for, and additional records on deal_sku_cost for each location, with location-specific discounts applied on top of the location-independent discounts.

- For each new LUW + loc, reset the flag for 'Fixed Amt value type discount. 'Fixed Amt value type discount should only be applied once for each LUW + loc.
- For each new LUW, reset the flag and merchandise level for 'Exclusive deal class discount; for each LUW + loc, reset the merchandise/organization level for 'Exclusive deal class discount (merchandise level needed to be reset back to before any loc-specific applied). 'Exclusive deal class discount should only be applied once for each LUW + loc.
- Reset the net/net net/dead net net costs according to the following rules:
- If new LUW, set to supplier's original unit cost
- If the same LUW, check if location changed:
- If new loc:
- Check if just change from loc-independent to loc-specific. If yes, save net/net net/dead net net costs and the applied merch level (for 'Exclusive discount) of loc-independent discounts
- Check if the flag for 'Exclusive deal class discount is set (previous 'Exclusive discount applied)
- If NO previous 'Exclusive discount applied, check if this is an 'Exclusive discount:
- If yes, set net/net net/dead net net costs to base cost (supplier's unit cost)
- If no, set net/net net/dead net net costs to costs of loc-independent discounts
- If previous 'Exclusive discount applied check if this is an 'Exclusive discount with higher merch level or equal merch level but higher org level than the saved merch/org level (only apply the highest merch/org level 'Exclusive discount):
- If yes, set net/net net/dead net net costs to base cost (supplier's unit cost)
- If no, skip this discount.
- If the same loc, check if the flag for 'Exclusive deal class discount is set (previous 'Exclusive discount applied)

- If NO previous 'EX'clusive discount applied, check if this is an 'EX'clusive discount:
- If yes, set net/net net/dead net net costs to base cost (supplier's unit cost)
- If no, set net/net net/dead net net costs to latest calculated costs
- If previous 'EX'clusive discount applied check if this is an 'EX'clusive discount with higher merch level or equal merch level but higher org level than the saved merch/org level (only apply the highest merch/org level 'EX' discount):
- If yes, set net/net net/dead net net costs to base cost (supplier's unit cost)
- If no, skip this discount.
- Call calculate_costs to calculate net/net net/dead net net costs. For the same LUW + loc, the driving cursor has sorted the discounts by cost_appl_ind: 'N' first, 'NN' later, 'DNN' last. For each cost application level, the same business rules are followed.
- If new LUW + loc, increment the writing index of the cost array (we always write a record into the cost array to keep track of last calculated costs, but change to a new record only if the LUW or the location is changed)
- Prepare a insert record into the deal_sku_cost table by writing costs into the current indexed record of the cost array. Use the start_date from deal_sku_temp as the active_date, and the unit_cost from item_supp_country as the base_cost. The location and location type fields should be left NULL if no location was given on deal_sku_temp. Vdate should be used for the calc_date.
- Save current processed LUW and loc.

calculate_costs:

Inputs: index of fetch array, target threshold value, current net/net net/dead net net costs

Outputs: calculated net/net net/dead net net costs

The definition of different net costs are:

net cost = unit cost – components whose cost_appl_ind is 'N'

net net cost = net cost – components whose cost_appl_ind is 'NN'

dead net cost = net net cost – components whose cost_appl_ind is 'DNN'

Use the cost_appl_ind on deal_detail to figure out whether a deal component contributes to the net, net net, or dead net net cost (the records should already be sorted by cost_appl_ind) and what the initial costs are (initial cost are need to process 'CU' mulative deal class discounts with 'P' ercentage value type):

- If 'N', the initial net cost is the supplier's original unit cost, and need to update all 3 net costs with the calculated discount
- If 'NN', the initial net net cost is the current net cost, and need to update both net net cost and dead net net cost with the calculated discount.
- If 'DNN', the initial dead net net cost is the current net net cost, and need to update only the dead net net cost with the calculated discount.

Business rules that need to be followed when applying discounts:

- Deal classes:

If an exclusive deal was previously found for this sku/supplier/origin country/start date: new cost should be calculated only if THIS deal is also exclusive and is for a lower merchandise hierarchy. If this is the first exclusive deal, process it and set a flag, saving the hierarchy levels.

Cumulative discounts need to be applied to the original unit cost (2% off + 3% off = 5 %off original unit cost)

Cascade discounts need to be applied on the result thus far ("current cost")--- take 2% off of the unit cost, then take 3% off of that price, for example

- Deal value types (take N cost calculation for example): for a % discount

If 'CS'cade:

discount cost = unit cost – (unit cost *%/100)

If 'CU' mulative:

discount cost = unit cost – (initial unit cost *%/100)

for an amt discount (first convert amount to be in supplier currency if necessary)

discount cost = unit cost – amt (amount discounts are per unit cost already)

fixed amt: if have fixed amount discount must start with THAT amount rather than the unit cost (convert to supplier currency if necessary)

discount cost = fixed amt (converted to supplier's currency if necessary)

- quantity discount ("buy some get some at discount") (these are not allowed on rebates)

These are the most complicated. They affect the cost of the get item AND of the buy item, whose cost we also need to get. Both the get item and the buy item will be on deal_itemloc. You should only calculate the cost for whichever item you're presently on (if buy item, just calculate buy item cost; will get the free item separately later, or vice versa). The initial unit cost for the get item should be taken from deal_detail.qty_thresh_free_item_unit_cost (or, if that field is not populated, off of item_supp_country). Before any calculations are done, convert the unit costs into supplier currency if necessary. If a buy/get free discount is encountered, the following things need to happen:

- Call get_unit_cost to get the original unit cost for the buy item (from item_supp_country), if it's different from the free item. Use the supplier and origin country of the free item (free and buy items are required to come from the same supplier and country).
- Calculate the discount costs(for whichever is the current item, free or buy)
- If qty_thresh_buy_target of the buy item < qty_thresh_buy_qty, stop; you didn't get any discount

Otherwise, figure out how many free items you actually get.

If the qty_thresh_recur_ind is 'N':

free qty = deal_detail.qty_thresh_free_qty

If the qty_thresh_recur_ind is 'Y':

If buy item = free item:

$$\text{free qty} = \text{qty\_thresh\_free\_qty} * \text{FLOOR}(\text{qty\_thresh\_buy\_target} / (\text{qty\_thresh\_buy\_qty} + \text{qty\_thresh\_free\_qty}))$$

- If buy item different from free item:

$$\text{free qty} = \text{FLOOR}(\text{qty\_thresh\_buy\_target} / \text{qty\_thresh\_buy\_qty}) * \text{qty\_thresh\_free\_qty}$$

If buy item = free item:

- If qty_thresh_get_type is 'X', this is a "buy/free" discount:

total discount = total get cost

- If qty_thresh_get_type is 'P', this is a "buy/get % off" discount:

$$\text{total discount} = (\text{get item's unit\_cost} * \text{qty\_thresh\_get\_value} / 100) * \text{get qty}$$

- If qty_thresh_get_type is 'A', this is a "buy/get amt off" discount:

$$\text{total discount} = \text{qty\_thresh\_get\_value} * \text{get qty}$$

- If qty_thresh_get_type is 'F', this is a "buy/get at fixed amt" discount:
$$\text{total discount} = (\text{get item's unit\_cost} - \text{qty\_thresh\_get\_value}) * \text{get qty}$$

- Discount rate = total discount / (buy item unit cost + buy target)

- Discount = discount rate * get item unit cost

If the free item and the buy item are different:

- If qty_thresh_get_type is 'X', this is a "buy/free" discount:

$$\text{total discount} = \text{total get cost}$$

- If qty_thresh_get_type is 'P', this is a "buy/get % off" discount:

$$\text{total discount} = (\text{get item's unit\_cost} * \text{qty\_thresh\_get\_value} / 100) * \text{get qty}$$

- If qty_thresh_get_type is 'A', this is a "buy/get amt off" discount:

$$\text{total discount} = \text{qty\_thresh\_get\_value} * \text{get qty}$$

- If qty_thresh_get_type is 'F', this is a "buy/get at fixed amt" discount:
$$\text{total discount} = (\text{get item's unit\_cost} - \text{qty\_thresh\_get\_value}) * \text{get qty}$$

- Get discount rate = (get item cost * get qty) / total buy cost

- Buy get discount rate = 1 – get discount rate

- If current item is buy item

$$\text{Discount} = \text{total discount} * \text{buy discount rate} / \text{buy target}$$

- If current item is get item

$$\text{Discount} = \text{total discount} * \text{get discount rate} / \text{get qty}$$

- If the total cost of the buy item is less than that of total discount, stop; no discount is applied

- These discounts are the amount that needs to be subtracted from the original price to get the discounted price.

`get_target_threshold_value:`

Given a `deal_id` and `deal_detail_id`, fetch the target value from the `deal_threshold` table (the value where the `target_id` is 'Y'). Since this function is often called multiple times for the same input (multiple SKUs of the same deal/deal detail), a linked list is maintained to keep track of target threshold values for different deal/deal detail. The linked list is ordered by the deal/deal detail. This function first tries to get the value from the list (previously fetched from database), if yes, job is done. Otherwise, fetch the target value for this deal/deal detail from database and call `convert_currency` if the value is currency amount and the deal currency is different from the supplier's currency. The newly fetched value is then saved into the list by calling `add_to_list`. Other maintenance functions for the linked list are `init_list` (called in `init`) and `free_list` (called in `final`).

`get_unit_cost:`

For a given `sku/supplier/country id`, get the unit cost from `item_supp_country`. Since usually the unit cost is fetched by the driving cursor, the function is only called for buy-get type discount when the buy item's unit cost is needed.

`convert_currency:`

Call `CURRENCY_SQL` package to convert an amount in deal currency to equivalent amount in supplier's currency. (This should only be called if the currencies are different—normally they will be the same).

`post_insert_delete_records:`

Array insert all records of the cost array into the `deal_sku_cost` table and array delete processed records, which are also all records of the cost array, from the `deal_sku_temp` table. This deletes all records from `deal_sku_temp` for a given `sku/supplier/origin country/start date/location`, the unique key of these five columns are part of the unique key on `deal_sku_cost`, which contains one more column (`calc_date`) to save the cost information for a system specified history month.

`add_to_list:`

Add a node made of deal/deal detail and the target value to the current position of the linked list.

`init_list:`

Initialize the linked list for target threshold values.

`free_list:`

Free the memory used by the linked list for target threshold values.

size_arrays:

Allocate memory for the fetch array used by the driving cursor and the cost array used to save the costs.

resize_array:

Allocate additional memory for the cost array.

free_arrays:

Free the memory used by the fetch array and cost array.

final:

- Call free_arrays and free_list.
- Restart/recovery close logic.

I/O Specification

N/A

[dealcls.pc]

Design Overview

This new batch program will close deals that have reached their end dates. This program will not cover PO-specific deals, which will be maintained with the PO.

Table	Select	Insert	Update	Delete
period	X			
deal_head	X		X	
Deal_queue		X		

Scheduling Constraints

Processing Cycle:	Phase 3 (daily)
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	NONE

Restart Recovery

Logical Unit of Work: deal_id

Driving Cursor:

```

SELECT deal_id,
       :ps_vdate
FROM deal_head
WHERE type != 'O'
      AND status = 'A'
      AND close_date <= TO_DATE(:ps_vdate, 'YYYYMMDD')
ORDER BY deal_id;
```

Since all the processing is on DEAL_HEAD, restart/recovery will be done naturally (driving cursor will not fetch updated records again) and thus no need to use restart variables. Restart/recovery is used here just for logging purposes.

New record for table RESTART_CONTROL:

```
INSERT INTO restart_control values ('dealcls', 'close deals that have reached  
close_date', 'NONE', 1, 'Y', 'T', 1000);
```

Note: The commit_max_ctr is assumed as 1000 here but it should be tuned at each client site.

Function Level Description

init:

- Fetch vdate in the format of 'YYYYMMDD' from table PERIOD.
- Initialize restart/recovery to get commit_max_ctr, driver_name, num_threads, and thread_val.

process:

- Array fetch the driver cursor with size of commit_max_ctr.
- Array update the table DEAL_HEAD with size as the number of records from each fetch of the driving cursor. Set status to 'C', close_date to vdate, and close_id to constant 'dealcls.pc'.
- Insert a record into the table DEAL_QUEUE for the fetched deal_id.
- Since the driving cursor fetch is of unique LUW, call restart/recovery force commit for each fetch.

final:

- Close restart/recovery.

Init_deal_head_array:

- Allocate memory for the driving cursor fetch array pa_deal_fetch.

Init_insert_array:

- Allocate memory for the array pa_deal_insert which will temporally hold the deal_id for deal_queue table.

Free_deal_head_array:

- Release memory for the array pa_deal_fetch.

Free_insert_array:

- Release memory for the array pa_deal_insert.

Prepare_insert_record:

- Fetch deal_id from the array pa_deal_fetch into pa_deal_insert array exclude those records existed in the table deal_queue. For a given deal, check to see if it exists on the deal_queue table. If it doesn't, put it in the array to be inserted.

Post_insert_records:

- Array insert deal_id from the array pa_deal_insert into the table deal_queue.

I/O Specification

N/A

[dealprg.pc]

Design Overview

This new batch program will purge deals after they have been held in the system for the specified number of history months after they are closed. This program will not cover PO-specific deals, which will be purged with the PO.

Table	Select	Insert	Update	Delete
Period	X			
system_options	X			
deal_head	X			X
deal_detail				X
deal_itemloc				X
deal_threshold				X
deal_queue				X

This program will be modified to exclude deals from purging if they exist on the ORDHEAD_DISCOUNT table.

Scheduling Constraints

Processing Cycle:	Ad Hoc
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

Logical Unit of Work: deal_id

Driving Cursor:

```
SELECT d.deal_id
FROM deal_head d
WHERE d.type != 'O'
AND d.status = 'C'
AND MONTHS_BETWEEN(TO_DATE(:ps_vdate,'YYYYMMDD'),
d.close_date) > :pi_deal_hist_months
AND NOT EXISTS (select 'x'
from ordsku_discount
where deal_id = d.deal_id)
ORDER BY deal_id;
```

Inherent restart/recovery: Since the processing is marked by deleting records from DEAL_HEAD, restart/recovery will be done naturally (driving cursor will not fetch deleted records again) and thus there is no need to use restart variables. Restart/recovery is used only for logging purposes.

New record for table RESTART_CONTROL:

```
INSERT INTO restart_control values ('dealprg', 'purge deals closed for a
system-specified period', 'NONE', 2, 'Y', 'T', 1000);
```

Note: The commit_max_ctr is assumed as 1000 but it should be tuned at each client site.

Function Level Description

init:

- Fetch vdate from table PERIOD.
- Fetch deal_history_months from table SYSTEM_OPTIONS.
- Initialize restart/recovery to get commit_max_ctr.

init_deals_array:

- Allocate memory to main fetching structure based on the dealprg's commit_max_ctr.

process:

- Array fetch the driver cursor with size of commit_max_ctr.
- Array delete from the following tables (in the order listed) with size as the number of records from each fetch of the driving cursor: DEAL_QUEUE, DEAL_ITEMLOC, DEAL_THRESHOLD, DEAL_DETAIL, DEAL_HEAD.
- Since the driving cursor fetch is of unique LUW, call restart/recovery force commit for each fetch.
- modify the driving cursor to also exclude deals that exist on the ORDHEAD_DISCOUNT table (similar to the exclusion of deals that exist on ORDSKU_DISCOUNT)

final:

- Close restart/recovery.

[ditinsrt.pc]

Design Overview

This new batch program will populate the deal_sku_temp table with all items that are on non vendor-funded, non PO-specific deals listed on the deal_queue table and all items that fall within a hierarchy from these deals. It will get values for the entire merchandise and organizational hierarchies to populate deal_sku_temp. The deal_sku_temp table will then be used by dealcalc.pc to recalculate costs for all listed items.

In addition, this program will populate the deal_calc_queue table with orders that may be affected by non vendor-funded, non PO-specific deals that are on the deal_queue table (for future processing by orddsent.pc). Orders that had been applied to deals that no longer apply will also be inserted into the deal_calc_queue table.

Scheduling Constraints

Processing Cycle: Phase I (daily)

Scheduling Diagram: This program must be run before dealcalc.pc and orddsent.pc (although it doesn't have to be run each time orddsent is run—once a night should be adequate),

Pre-Processing:

Post-Processing:

Threading Scheme: None—this program is not threadable (can't thread by supplier because we may not have a supplier on deal_head)

Restart Recovery

The logical unit of work is the deal.

SELECT di.merch_level,

di.division,

di.group_no,

di.dept,

di.class,

di.subclass,

```
        di.style,
        di.color,
        di.size1,
        di.size2,
        NVL(di.org_level,0),
        di.chain,
        di.area,
        di.region,
        di.district,
        di.loc_type,
        di.location,
        di.sku,
        di.origin_country_id,
        dh.supplier,
        dh.partner_type,
        dh.partner_id,
        TO_CHAR(dh.active_date,'YYYYMMDD'),
        TO_CHAR(dh.close_date,'YYYYMMDD'),
        dh.status,
        dh.recalc_approved_orders,
        dq.deal_id,
        di.deal_detail_id

FROM   deal_queue dq, deal_itemloc di, deal_head dh
WHERE  dh.deal_id = dq.deal_id

      AND dh.deal_id = di.deal_id

      AND dh.type <> 'M'           /* no vendor-funded-markdown deals */
```



```

AND dh.type <> 'O' /* no order-specific deals */

AND di.excl_ind = 'N'

AND dq.deal_id > NVL(:ps_restart_deal, -999)

ORDER BY dq.deal_id, di.deal_detail_id;

A commit is forced for each new deal.

```

Program Flow

Tables used:

Table	Select	Insert	Update	Delete
deal_itemloc	X			
deal_head	X			
item_supp_country	X			
deal_sku_temp		X		
deal_queue	X			X
desc_look	X			
packhead	X			
chain	X			
area	X			
region	X			
district	X			
store	X			
division	X			
groups	X			
deps	X			
rag_skus	X			
deal_calc_queue		X		
ordhead	X			
ordsku_discount	X			

Shared Modules

N/A

Function Level Description

init:

Restart/recovery initialization

process:

Fetch the driving cursor (use an array fetch into an array of structs and use indicator variables—on deal_itemloc, only merch_level is guaranteed not to be NULL)

For each different deal (each unique deal_id), call get_orders to do the necessary inserts into the deal_calc_queue table (which holds orders that may be affected by deals).

For each row that the fetch returned (each different deal_itemloc record), the values that were not populated on deal_itemloc (everything below the particular hierarchy level given) must now be obtained as follows:

1. Call explode_merch to find all additional merchandise hierarchy and origin country information, creating an array of records that contain all the information passed in plus merchandise hierarchy and origin countries. Also find supplier information if the deal has a partner rather than a supplier.
2. Call explode_org to find all additional organizational hierarchy information, creating the final array that will be inserted into deal_sku_temp.

Once all item records have been fully exploded for a deal, they should be inserted into the deal_sku_temp table (call insert_items), the deal should be deleted from deal_queue (call delete_dq), and a forced commit should take place.

explode_merch:

For each record from the deal_itemloc arrays, we need to obtain all missing merchandise hierarchy and origin country and supplier information, as follows:

(whenever getting skus, make sure to outer join to rag_skus so as to get staple and pack skus also. Only vendor pack items should be used—skip buyer packs, using an outer join to packhead))

- If merch_level = 10 get all origin countries for this sku/supplier from item_supp_country if an origin country was not given. (We should already have all necessary merchandise hierarchy information). If no supplier was given, we must get all suppliers on item_supp_country that have the same partner as the deal (if the partner_id on deal_head is 'S1', get all suppliers from item_supp_country where supp_hier_lvl_1 matches the partner id; if it's 'S2', where supp_hier_lvl_2 matches, if 'S3', where supp_hier_lvl_3 matches. If the partner_id is 'S', a supplier is given on deal_head; just match to that supplier. This supplier match must be done for all merch levels.
- If merch_level = 9 (have info down to style/size), need to get all skus for the supplier/style/size1/size2 from rag_skus and item_supp_country, and all origin countries from item_supp_country if the origin country was not given. (Remember that size1 and size2 might both be given, or either of them could be NULL).
- If merch_level = 8 (info down to style/color), need to get all skus (and color and size information) for this supplier/style/color from rag_skus and item_supp_country, and all origin countries (if the origin country was not given) from item_supp_country.
- If merch_level = 7 (info down to style level), get all skus for this supplier and style, along with color and size information and all origin countries (if the origin country was not given) from rag_skus and item_supp_country.
- If merch_level = 6 (info down to subclass level), get all sku and style/color/size info for this supplier from item_supp_country, desc_look and rag_skus and all origin countries (if the origin country was not given) from item_supp_country. Don't get skus on desc_look whose system_ind = 'F', since they are really styles.
- If merch_level = 5 (info down to class level), get all subclass/skus/color/style/size information info for this supplier from item_supp_country, desc_look and rag_skus and all origin countries (if the origin country was not given) from item_supp_country. Don't get skus on desc_look whose system_ind = 'F', since they are really styles.
- If merch_level = 4 (info to dept level), get all class/subclass/sku/color/style/size information info for this supplier from item_supp_country, desc_look and rag_skus and all origin countries (if the origin country was not given) from item_supp_country. Don't get skus on desc_look whose system_ind = 'F', since they are really styles.

- If merch_level = 3 (info to group level), get all departments for the group from deps, get class/subclasses/skus/colors/styles/sizes for each department info for this supplier from item_supp_country, desc_look and rag_skus and all origin countries (if the origin country was not given) from item_supp_country. Don't get skus on desc_look whose system_ind = 'F', since they are really styles.
- If merch_level = 2 (info to division level) get all groups under that division from groups, and all departments under each group from deps. Then get all skus and their style and color and sizes (if fashion) for this supplier from item_supp_country, desc_look, and rag_skus and all origin countries (if the origin country was not given) from item_supp_country for that dept (where system_ind != F—don't want to get styles from desc_look)
- If merch_level = 1 (Company level--ALL items for a supplier), get all divisions from the division table, get all groups for each division from groups, and all departments under each group from deps. Then get all skus and their style and color and sizes (if fashion) info for this supplier from item_supp_country, desc_look, and rag_skus and all origin countries (if the origin country was not given) from item_supp_country for that dept (where system_ind != F—don't want to get styles from desc_look)

Use subfunctions to get as much of the repeated stuff as possible (call get_dept_info to get the department-level stuff, on down).

This will probably be set up as a big case statement; each merch level value calls a different function

As the information is obtained, it should be inserted into the merchandise array (use copy_deal_array). Be sure to resize this array before inserting, if necessary. (Note that the explosion process will potentially create far more records than were passed in--if only given a department, a record needs to be created for every sku in that department, for example).

explode_org:

For each record from the merchandise arrays (containing merchandise/country information and all organization information that was present on deal_itemloc), get any additional organizational hierarchy information as follows:

- If nothing is given in the organizational hierarchy (org_level is NVL'd to 0), leave the chain, area, district, region, location, location type fields NULL when inserting into deal_sku_temp (no additional information needs to be fetched)
- If the org_level = 5 (location) and the given location is a warehouse, the chain, area, region, and district fields should be left as NULL when inserted into deal_sku_temp—warehouses aren't part of a district (no additional information needs to be fetched)

- If org_level = 5 (location) and the given location is a store, all necessary information should already be present; no additional information needs to be fetched.
- If org_level = 4 (district) , get all stores in this district from the store table
- If org_level = 3 (region), get all districts and stores in this region from the district and store tables
- If org_level =2 (area), get all regions, districts, stores for this area from region, district, and store tables
- If org_level = 1 (chain), get all areas, all regions, all districts, all stores for this chain from the area, region, district, and store tables

Keep track of how many organizational hierarchy combinations were fetched and for what level so that if the next record has the same organization level for the same organization hierarchy element (say, it's org level 4 and the current district is the same as the last district) the organization values can just be copied from the ones previously obtained instead of having to get them from the database again.

As the information is obtained, it should be inserted into the final array (use copy_deal_array) for insert to deal_sku_temp (be sure to resize this array if necessary).

Get_origin_country:

Given an array containing the records retrieved by the driving cursor, find all countries and suppliers for the given item, and copy the info into the passed output array. This function should only be called if all merch info is already known (merch_level 10), except for the country.

Get_sku_info:

Given an array containing the records retrieved by the driving cursor, find all missing merch information, and copy the info into the passed output array. This function should only be called if style is not null but sku is (merch_levels 7-9).

Get_dept_info:

Given an array containing the records retrieved by the driving cursor, find all missing merch information and copy the info into the passed output array. This function should only be called if dept is not null but style is (merch_levels 4-6).

Get_group_info:

Given an array containing the records retrieved by the driving cursor, find all missing merch information and copy the info into the passed output array. This function should only be called if group_no is not null but dept is (merch_level 3).

Get_division_info:

Given an array containing the records retrieved by the driving cursor, find all missing merch information and copy the info into the passed output array. This function should only be called if division is not null but group_no is (merch_level 2).

Get_company_merch_info:

Given an array containing the records retrieved by the driving cursor, find all merch information and copy the info into the passed output array. This function should only be called if no merch information is given (merch_level 1).

Get_stores_districts_regions_areas

Given a specific deal in an array containing all items from the driving cursor with all merch information already filled in, find all locations existing under the given chain and copy them to the output array. Function also takes in another deal array and an indicator variable. If the indicator is true, then use org information contained in this third array rather than running queries.

Get_stores_districts_regions

Given a specific deal in an array containing all items from the driving cursor with all merch information already filled in, find all locations existing under the given area and copy them to the output array. Function also takes in another deal array and an indicator variable. If the indicator is true, then use org information contained in this third array rather than running queries.

Get_stores_districts

Given a specific deal in an array containing all items from the driving cursor with all merch information already filled in, find all locations existing under the given region and copy them to the output array. Function also takes in another deal array and an indicator variable. If the indicator is true, then use org information contained in this third array rather than running queries.

Get_stores

Given a specific deal in an array containing all items from the driving cursor with all merch information already filled in, find all locations existing under the given district and copy them to the output array. Function also takes in another deal array and an indicator variable. If the indicator is true, then use org information contained in this third array rather than running queries.

Copy_merch_info:

This function takes three deal record arrays and an index. The first array contains the information gathered from the driving cursor. The second contains the information gathered from the preceding get_<level>_info functions. For each record in the second array, copy all merch related information from the second array to the first array. Then insert the combined record into the third array using copy_deal_array.

Initialize_deal_info:

Initializes all indicator variables and sets deal record count to 0 for a passed deal record.

insert_items:

Array insert records into the deal_sku_temp table, first checking for duplicates (check to see if each record exists; if it doesn't put it into an array for insert; insert once all have been checked). This needs to be done twice, once with a start date of deal_head.active_date and once with a start_date of deal_head.close_date (if the close date is null, don't insert the record). (The duplicate check needs to be done both times).

Start_date_in_deal_sku_temp:

Check if insert record already exists in deal_sku_temp with deal_sku_temp.start_date = active_date.

End_date_in_deal_sku_temp

Check if insert record already exists in deal_sku_temp with deal_sku_temp.start_date = close_date.

Insert_into_dst_start_date:

Insert passed record into deal_sku_temp with deal_sku_temp.start_date = active_date.

Insert_into_dst_end_date:

Insert passed record into deal_sku_temp with deal_sku_temp.start_date = close_date.

Ord_exists

Sets return_flag to true if passed ord is already in passed ord_list.

Copy_ord_to_final

This function takes an output array, an array containing all merch information, and an array containing org information. For each record in the org array, copy information to given record in the merch array, and place this combined record into the output array.

Copy_ord_numbers

Given to ord_lists and an index, copy value at index from start array to final array if the value does not already exists in the final array.

In_deal_calc_queue:

Set return flag to true if passed ord_num is already in deal_calc_queue

Insert_deal_calc_queue:

Insert values contained in given ord_list into deal_calc_queue.

Initialize_ord_list:

Assign memory for ord_list, and set ord_list count to 0.

Resize_ord_list

Allocate additional memory for deal array. Allocation increments by commit_max_ctr.

get_orders:

Given a deal id, a deal status, the deal's recalc_approved_orders indicator, and the deal start (active) and close dates,

- If the deal's recalc_approved_orders is 'Y', call insert_approved if the status is 'A'pproved,
- Call insert_unapproved if the status is 'W'orksheet (it was unapproved and just got set back to worksheet status)
- Call insert_closed if the status is 'C'losed.

This will populate the deal_calc_queue table with orders that may be affected by the deal.

insert_approved:

Select all approved orders from the ordhead table whose not before date is between the deal start date and the deal close date (or just after the deal start date, if the deal close date is NULL). Insert these orders into the deal_calc_queue table (recalc_all_ind, order_appr_ind, and override_manual_ind should be inserted as 'N').

insert_unapproved:

Select all orders from the ordsku_discount table for the given deal that do not have a status (on ordhead) of 'C'losed, and insert these orders into the deal_calc_queue table (recalc_all_ind, order_appr_ind , and override_manual_ind should be inserted as 'N').

insert_closed:

Select all orders from the ordsku_discount table for the given deal whose not before date is after the deal's close date (all closed deals must have a close date) and whose status (from ordhead) is not 'C'losed. Insert these orders into the deal_calc_queue table (recalc_all_ind, order_appr_ind , and override_manual_ind should be inserted as 'N').

select distinct od.order_no

from ordsku_discount od, ordhead oh

where od.order_no = oh.order_no

and od.deal_id = :is_deal_id

and oh.not_before_date > to_date(:is_close_date,'YYYYMMDD')

get_orders_with_altered_deals:

Select all orders from the ordsku_discount table for the given deal whose not before date is after the deal's close date regardless of the status. Insert these orders into the deal_calc_queue table (recalc_all_ind and override_manual_ind should both be inserted as 'N', order_appr_ind should be 'Y')

size_array

Allocate memory for passed array.

resize_deal_array

Allocate additional memory for deal array. Allocation increments by commit_max_ctr.

Copy_deal_array:

Given two deal arrays and an index, copy deal at index from I_deal_info to o_unique_records. This function will resize o_unique_records if necessary, but it will not check if deal already exists in the final array.

delete_dq:

Delete processed records from the deal_queue table

final:

Restart/recovery close logic

I/O Specification

N/A

Daily Record Deletion [dlyprg]

Design Overview

The purpose of this program is to delete all of the records in the system marked for delete (by having a record on the DAILY_PURGE table) during the day. Before deleting the records, all relations will be checked to ensure that the record can be deleted. For example, if a staple SKU has been marked for delete, this program will check that the SKU was not put on order later in the day. If relations are found to exist, a record is written to the DAILY_PURGE_ERROR_LOG table. Records on this table will be used to generate a report itemizing any problems found when running this program. If a record is written to the DAILY_PURGE_ERROR_LOG table, meaning that relations exist, the record will not be deleted that night.

After removing records marked for deletion, the program will delete from tables (i.e. MASK_TEMP) that must be cleared out nightly for maintenance reasons.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DAILY_PURGE	No	Yes	No	Yes	No
DAILY_PURGE_ERROR_LOG	No	Yes	No	Yes	Yes
MASK_TEMP	No	No	No	No	Yes

Scheduling Constraints

Processing Cycle: PHASE 0 (daily)

Scheduling Diagram: This program must run first to avoid processing deleted entities.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

```
EXEC SQL DECLARE c_daily_purge CURSOR FOR  
    SELECT      key_value,  
               table_name,  
               delete_type  
    FROM daily_purge  
    ORDER BY    delete_order;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Increment Set System Date [dtesys]

Design Overview

Gets a date from the script in run_date() and builds a new row in the PERIOD table after deleting any existing row.

The flag, which indicates that the system is available to users, is maintained here.

Daily:

If user provides a second command line argument, it is used as the new vdate. Otherwise, vdate will increment by 1.

Weekly:

When vdate = next_eow_date_unit, update the SYSTEM_VARIABLES columns last_eow_date_unit and next_eow_date_unit, setting last_eow_date_unit to equal next_eow_date_unit and next_eow_date_unit to equal the next end-of-week date (calculated).

Monthly:

When vdate = next_eom_date_unit, update the SYSTEM_VARIABLES columns last_eom_date_unit and next_eom_date_unit, setting last_eom_date_unit to equal next_eom_date_unit and next_eom_date_unit to equal the next end-of-month date (calculated).

Tables affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	Yes	No	Yes
SYSTEM_OPTIONS	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	Yes	No

Scheduling Constraints

Processing Cycle: PHASE 4 (daily)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Retek Store and Warehouse Address Download [edidladd]

Design Overview

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

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

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

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

Indexes: store (store)

wh (wh)

Scheduling Constraints

Processing Cycle: PHASE 4 (DAILY)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

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

Driving cursor:

```
SELECT :ps_str_type,
       store.store,
       store.store_name,
       store.store_add1,
       store.store_add2,
       store.store_city,
       store.state,
       store.store_pcode,
       store.country_id,
       store.addr_chg,
       ROWIDTOCHAR(rowid)
FROM store
WHERE store.addr_chg = DECODE(:pi_do_all,1,store.addr_chg,'Y')
UNION ALL
SELECT :ps_wh_type,
       wh.wh,
       wh.wh_name,
       wh.wh_add1,
       wh.wh_add2,
       wh.wh_city,
```



```
        wh.state,  
        wh.wh_pcode,  
        wh.country_id,  
        wh.addr_chg,  
        ROWIDTOCHAR(rowid)  
FROM wh  
  
WHERE wh.addr_chg = DECODE(:pi_do_all,1,wh.addr_chg,'Y');
```

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

Program Flow

N/A

Shared Modules

N/A

Function Level Description

Init()

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

Process()

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

Write FTAIL record to file.

Write_records_loc()

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

Write_records_cat()

This function writes address change records to the catalog file.

Final()

Call restart/recovery close.

I/O Specification

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

Record Name	Field Name	Field Type	Default Value	Description
File header	File type record descriptor	Char(5)	FHEAD	Identifies file record type
	File line sequence	Number(10)	Start at 1 and increment	Line number of file
	File type definition	Char(5)	DLADD	Identifies file source
	Purpose code	Char(2)	04 (change) or 05 (replace)	Add/change location or replace whole list
Transaction Detail	File type record descriptor	Char(5)	TDETL	Identifies file record type
	File line sequence	Number(10)	Increment	Line number of file
	Transaction number	Number(10)	Start at 1, increment	Identifies transaction
	Date	Char(8)		Period.vdate YYYYMMDD
	Store or warehouse	Char(2)	ST (store) or WH (warehouse)	Location type
	Location	Number(4)		Store.store or wh.wh
	Location name	Char(20)		Store..store_name or wh.wh_name
	Address line 1	Char(30)		Store.store_add1/wh .wh_add1
	Address line 2	Char(30)		Store_add2 or wh_add2
	City	Char(20)		Store.store_city/wh. wh_city
	State	Char(3)		Store.state/wh.state

Record Name	Field Name	Field Type	Default Value	Description
	Postal code	Char(10)		Store_pcode/wh_pcode
	Country	Char(3)		Store/wh.country_id
File trailer	File type record descriptor	Char(5)	FTAIL	Identifies file record type
	Total number lines	Number(10)		Total lines in file
	Total no. TDETL lines	Number(10)		(total lines – 2)

EDI Contract Information Download [edidlcon]

Design Overview

This program downloads a file of EDI contract information. Contracts are only processed if they are in approved status and have an edi_contract_ind of 'Y'.

Changes to make: add restart recovery and make output ONE file instead of one per supplier. Minor changes to file format (add Gentran ID in FHEAD line and move supplier from FHEAD to THEAD; include transaction number on transaction lines). Let the user enter an output file name.

Scheduling Constraints

Processing Cycle:	Daily, Phase 4
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A –file based processing

Restart Recovery

```
SELECT ch.contract_no,
       ch.contract_type,
       ch.dept,
       TO_CHAR(ch.supplier),
       TO_CHAR(NVL(ch.total_cost*1000,0)),
       ch.edi_contract_ind,
       ch.currency_code,
       ROWIDTOCHAR(ch.rowid)
FROM   contract_header ch
WHERE  ch.status = 'A'
       AND ch.edi_contract_ind = 'Y'
       AND ch.edi_sent_ind = 'N'
ORDER BY ch.supplier
```

Program Flow

N/A

Shared Modules

CONTRACT_SQL.GET_UNIT_COST – get the cost of a contract-sku.

Function Level Description

Include the std_rest.h library.

Init:

Get period.vdate.

Call restart_file_init

Make the format strings for output file lines

Open output file and write fhead line with the write_head function

Process:

Fetch contract header cursor

Update contract_header.edi_sent_ind to 'Y'

write information out to file (call write_detail and write_summary)

use restart_commit for commits

write_head

write FHEAD line to file

write_detail

For contract types C and D (no production plans): Get sku contract info from the contract_detail table. Get upc info from upc_ean and upc_supplier tables. Write TDETL lines to file. For contract types A and B (production plans): Get sku contract info from the contract_detail table. Get upc info from the upc_ean and upc_supplier tables. Write TDETL lines to file. (Note: ready_date, ready_quantity, location_type, and location_number will only have values for contract types A and B).

write_summary:

write TTAIL string to file

write_trailer

write FTAIL string to file

Final:

Write output to final file and close files.

Call restart_close.

I/O Specification

Output File layout:

- FHEAD File identification and date
- THEAD Supplier and contract header info
- TDETL Item information
- TTAIL Transaction trailer
- FTAIL File trailer; total number of transactions written

All character variables should be right-padded with blanks and left_justified; all numerical variables should be left-padded with zeroes and right-justified. All dates should be in YYYYMMDDHH24MISS format.

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record descriptor	Char(5)	FHEAD	Describes file line type
	Line number	Number(10)	0000000001	Sequential file line number
	Gentran ID	Char(5)	dnent	Identifies transaction type for Gentran
	Current date	Char(14)		Current date period.vdate
THEAD	Record descriptor	Char(5)	THEAD	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Supplier	Char(10)		Contract_header.supplier
	Contract	No number (6)		Contract_header.contract_no
	Contract type	Char(1)		Contract_header.contract_type
	Department	Number(4)		Contract_header.dept
	Currency code	Char(3)		Contract_header.currency_code
	Total contract cost	Char(20)		Contract_header.total-cost*10000

Record Name	Field Name	Field Type	Default Value	Description
TDETL	Record descriptor	Char(5)	TDETL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Item type indicator	Char(3)		'SKU' or 'UPC'
	Item number	Char(13)		UPC number (SKU number if no UPC)
	UPC supplement	Number(5)		UPC supplement (zero if none)
	Color	Char(24)		Color (if this is a fashion item with color information)
	Size	Char(61)		Size (if this is a fashion item with size information)
	VPN	Char(30)		Vendor Product Number for a fashion item
	Unit cost	Char(2)		Contract_skku.unit_cost*10000 (4 implied decimal places)
The following variables will only have values for contract types of 'A' or 'B';				
	Ready date	Char(14)		Contract_prod_plan.ready_date
	Ready quantity	Char(20)		Contract_prod_plan.qty_ready*10000 (4 implied decimal places)
	Location type	Char(2)		'ST' or 'WH'
	Location number	Char(4)		Contract_prod_plan.store or .wh
FTAIL	Record descriptor	Char(5)	TTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number

Record Name	Field Name	Field Type	Default Value	Description
FTAIL	Record descriptor	Char(5)	FTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number (total # lines in file)
	Contract count	Number(10)		Total number of transactions in file

Technical Issues

N/A

[edidldeb.pc]

Design Overview

This program will download debit memo, credit note request, and credit memo data from the invoice matching tables into an ASCII file that will be converted into EDI format by the client and sent via EDI.

Scheduling Constraints

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
SYSTEM_OPTIONS		Yes	No	No	No
PERIOD		Yes	No	No	No
SUPS		Yes	No	No	No
INVC_HEAD		Yes	No	Yes	No
INVC_DETAIL		Yes	No	No	No
INVC_MERCH_VAT		Yes	No	No	No
INVC_XREF		Yes	No	No	No

Processing Cycle: Phase 4 (daily)

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: no threading

Restart Recovery

The logical unit of work is the invoice.

Driving cursor:

```
SELECT invc_id,
       invc_type,
       partner_type,
       partner_id,
       supplier,
       ext_ref_no,
       ref_invc_id,
       ref_rtv_order_no,
       ref_rsn_code,
       terms,
       TO_CHAR(due_date, 'YYYYMMDD')
       payment_method,
       TO_CHAR(invc_date, 'YYYYMMDD')
       currency_code,
       (exchange_rate * 10000000000),
       (total_merch_cost * 10000),
       (total_qty * 10000,
       ROWIDTOCHAR(ih.rowid)
FROM invc_head
WHERE status = 'A'
       AND edi_sent_ind = 'N'
       AND invc_type in ('D','R','M')
ORDER BY invc_id;
```

Implicitly restartable because the edi_sent_ind gets updated in the program.

Commits should occur every 10,000 records (subject to experimentation)

Shared Modules

N/A

Function Level Description

init()

restart/recovery initialization

Fetch vdate from period and vat_ind from system_options.

Open temporary output file and write FHEAD line (if program is not being restarted)

Set up format strings for output file writes

Call size_arrays to allocate memory for the arrays to be fetched by the driving cursor.

process()

Fetch the header information (user array fetches of the driving cursor)

If a cross-reference invoice id is given, call get_order to get the order numbers related to the invoice, if any.

For each credit request/memo or debit memo, write transaction header line to file (call write_THEAD)

For each credit request/memo or debit memo, call get_total_vat to retrieve totals at each different vat rate on the invoice, if any, and write TVATS lines.. Only do this if VAT is turned on in the system and a cross-reference invoice id is given.

For each credit request/memo or debit memo, fetch all related detail records and write those to the file by calling get_items.

For each credit request/memo or debit memo, write transaction trailer to file (call write_TTAIL)

Update the edi_sent_ind on the invc_head table to 'Y' (use the rowid for the update)

get_items()

For the given invoice id, fetch all records from the invc_detail table, format, and write to output file by calling write_TDETL. Each invc_detail record corresponds to one TDETL line.

write_THEAD()

Write invoice header information line to output file.

write_TVATS()

Write total amount at a given VAT rate information line to output file.

write_TDETL()

Write an item detail line to output file.

write_TTAIL()

Write transaction tail line to output file.

size_arrays()

Use calloc to reserve memory for the arrays to be fetched.

get_total_vat()

See if there are summary-level totals with associated vat rate that apply to the credit memo request/credit memo or debit memo. To do this, check to see if there are records on invc_merch_vat with an invc_id of the invoice that the debit memo or credit request refers to.

get_order()

See if there are order numbers associated with the invoice that the debit memo/credit request refers to by checking the invc_xref table. Fetch up to 5 of these (we assume that there will never be more than five different associated orders)

final()

Write FTAIL line to file

Restart/recovery close

Write temporary file to final file; remove temporary file; close final file.

I/O Specification

Output file format:

(There will be no TVATS lines if the system does not use VAT or if there are no records on invc_merch_vat; there can be several TVATS lines for each invoice. It is possible to have no TDETL lines or many TDETL lines for each invoice.)

FHEAD (start of file)

THEAD (Credit memo request/debit memo header level info)

TVATS (info from invc_merch_vat table—not present if VAT is turned off)

TDETL (items within a credit request/debit memo)

TTAIL (marks end of credit request/debit memo)

FTAIL (marks end of file)

All character variables should be right-padded with blanks and left justified; all numerical variables should be left-padded with zeroes and right-justified. Missing variables should be blank.

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	File head descriptor	Char(5)	FHEAD	Describes file line type
	Line id	Char(10)	0000000001	Sequential file line number
	Gentran ID	Char(5)	DLINV	Identifies which translation Gentran uses
	Current date	Char(14)		YYYYMMDDH H24MISS format (period.vdate)
THEAD	File record descriptor	Char(5)	THEAD	Start of an invoice transaction
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Transaction type flag	Char(1)		D=debit memo R=Credit note request M=Credit memo (invc_head.invc_type)

Record Name	Field Name	Field Type	Default Value	Description
	Invoice id	Number(10)		ID for the credit request or debit memo (invc_head.invc_id)
	Supplier invoice id	Char(30)		Supplier's id for credit/debit invc_head.ext_ref_no (blank if none)
	Cross-reference invoice number	Number(10)		Invoice that a credit note request/debit memo refers to invc_head.ref_invc_id (blank if none)
	Cross-reference RTV order number	Number(6)		Related RTV order number invc_head.ref_rtv_order_no (blank if none)
	Reason code	Char(6)		invc_head.ref_rsn_code (blank if none)
	Partner Type	Char(6)		Debit/credit partner's type invc_head.partner_type (blank if none)
	Partner	Char(10)		Debit/credit partner invc_head.partner_id (blank if none)
	Supplier	Number(10)		Invoice supplier invc_head.supplier

Record Name	Field Name	Field Type	Default Value	Description
	invoice date	Char(14)		Invc_head.invc_date
	Due date	Char(14)		invc_head.due_date YYYYMMDDH H24MISS
	Order number 1	Number(8)		Order that original invoice was for (if any known—from invc_xref)
	Order number 2	Number(8)		second order associated w/invoice if any
	Order number 3	Number(8)		third, if any
	Order number 4	Number(8)		fourth, if any
	Order number 5	Number(8)		fifth, if any
	Terms	Char(15)		Invc_head.terms
	Payment method	Char(6)		Invc_head.payment_method LC=letter of credit OA= Open Account WT= wire transfer
	Currency code	Char(3)		Currency code of invoice currency from invc_head.currency_code

Record Name	Field Name	Field Type	Default Value	Description
	Exchange rate	Number(20)		From the invoice currency to the primary currency. 10 implied decimal places.
	Total credit or debit merchandise cost	Number(20)		Total cost of all items that are part of the credit or debit (4 implied decimal places) invc_head.total_merch_cost
	Total quantity	Number(12)		Total quantity of items on credit or debit (4 implied decimal places) invc_head.total_qty Blank if none
TVATS	File record descriptor	Char(5)	TVATS	Marks total cost at given VAT rate line
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	VAT code	char(6)		invc_merch.vat_code
	cost excluding VAT	Number(20)		invc_merch_vat.total_cost_excl_vat
TDETL	File record descriptor	Char(5)	TDETL	Marks item info line
	Line id	Char(10)		Sequential file line number

Record Name	Field Name	Field Type	Default Value	Description
	Transaction number	Number(10)		Sequential transaction number
	SKU	Number(8)		Sku invc_detail.sku (blank if none)
	UPC	Char(13)		Upc (invc_detail.upc) blank if none
	UPC supplement	Number(5)		Upc supplement invc_detail.upc_s upplement (blank if none)
	Invoice quantity	Number(12)		How many of this item on credit request/debit memo (4 implied decimal places) invc_detail.invc_qty
	Unit cost	Number(20)		Unit cost of item 4 implied decimal places invc_detail.invc_unit_cost
MAY CHANGE to VAT CODE	VAT rate	Number(20)		VAT rate for this item (10 implied decimal places) (if exists, else blank) invc_detail.vat_rate
	Cost discrepancy indicator	char(1)		invc_detail.cost_dscrpnncy_ind
	Quantity discrepancy indicator	char(1)		invc_detail.qty_dscrpnncy_ind

Record Name	Field Name	Field Type	Default Value	Description
	VAT discrepancy indicator	char(1)		invc_detail.vat_discrepancy_ind
TTAIL	File record descriptor	Char(5)	TTAIL	Marks end of transaction
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Transaction lines	Number(6)		Total number of TDETL and TVATS lines in transaction
FTAIL	File record descriptor	Char(5)	FTAIL	Marks end of file
	Line id	Char(10)		Sequential file line number
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

New and Changed P.O. Download [edidlord]

Design Overview

Orders generated within the Retek system are written to a flat file if they are approved and specified as EDI orders. If shipments are to be pre-marked for cross-dock allocation by the supplier, allocation location and quantities will be sent along with the order information. If the order contains pack items, hierarchical pack information will be sent (this may include outer packs, inner packs, and fashion styles with associated pack templates, as well as component sku information). File output is to a Retek standard format file, with the translation to EDI format taking place via an outside translator such as Gentran.

Currently, edidlnew downloads new orders to an output file, while edidlchg downloads changed orders. These programs will be combined and modified to work with changes that have been made to the ordering tables. The order revision tables and allocation revision table will also be used, to ensure that the latest changes are being sent and to allow both original and modified values to be sent. These revision tables are populated during the online ordering process and the batch replenishment process whenever an order has been approved, and constitute a history of all revisions to the order.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ORDHEAD_REV	Yes	Yes	No	No	No
ORDHEAD	Yes	No	No	Yes	No
ORDSKU_REV	Yes	Yes	No	No	No
ORDLOC_REV	Yes	Yes	No	No	No
ITEM_SUPPLIER	No	Yes	No	No	No
UPC_EAN	Yes	Yes	No	No	No
UPC_SUPPLIER	Yes	Yes	No	No	No
ALLOC_HEADER	Yes	No	No	No	No
ALLOC_DETAIL		Yes	No	No	No
ALLOC_DETAIL_REV		Yes	No	No	No
DESC_LOOK		Yes	No	No	No
PACKSKU		Yes	No	No	No
SUPS_PACK_TEMP L_DESC		Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE 4 (may also be schedule ad hoc to run multiple times per day)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

Note: the starred (**) variables may be null, so they must be selected using an indicator variable and then have a value assigned if null.

```

SELECT
    ROWIDTOCHAR(oh.rowid),
    oh.order_no,
    to_char(oh.supplier),
    to_char(oh.written_date,'YYYYMMDDHH24MISS'),
    ** to_char(ohr.not_before_date,'YYYYMMDDHH24MISS') ←old not
before
date
    to_char(oh.not_before_date,'YYYYMMDDHH24MISS'), ←new not before
date
    ** to_char(ohr.not_after_date,'YYYYMMDDHH24MISS'), ←old
    to_char(oh.not_after_date,'YYYYMMDDHH24MISS'), ← new
    ** oh.vendor_order_no,
    ** ohr.currency_code, ← old
    oh.currency_code, ← new
    ** ohr.ship_pay_method, ← old
    ** oh.ship_pay_method, ← new

```

```
** ohr.fob_trans_res, < old
** oh.fob_trans_res, < new
** ohr.fob_trans_res_desc, < old
** oh.fob_trans_res_desc, < new
** ohr.fob_title_pass, < old
** oh.fob_title_pass, < new
** ohr.fob_title_pass_desc, < old
** oh.fob_title_pass_desc, < new

    oh.pre_mark_ind
    oh.last_sent_rev_no

FROM ordhead oh,
    ordhead_rev ohr,
    sups s,
    v_restart_supplier v
WHERE ohr.order_no (+) = oh.order_no
    AND oh.status = 'A'
    AND oh.edi_sent_ind = 'N'
    AND oh.edi_po_ind = 'Y'
    AND oh.supplier = s.supplier
    AND (s.edi_po_chg = 'Y'
        OR (s.edi_po_chg = 'N'
            AND oh.last_sent_rev_no IS NULL))
    AND ohr.origin_type (+) = 'V'
    AND ohr.rev_no(+) = oh.last_sent_rev_no
    and v.driver_name = :os_restart_driver_name
```

```

and v.driver_value = oh.supplier

and v.num_threads = :oi_restart_num_threads

and v.thread_val = :oi_restart_thread_val

ORDER BY 2 , 3;

```

Restart/recovery capability will be used in this program to provide restart capability. Restartability is implied because the program updates ordhead.edi_sent_ind as records are written out.

Program Flow

Orders that are in approved status and that are new or have had changes made are fetched from the system tables. Additional information about the items on the order and their destination is gathered. Order, item, pack, and shipment information is written to an output file. The system tables are then updated to show that the orders have been sent.

Shared Modules

N/A

Function Level Description

init()

Get current date. Set up format strings for output file, open output file, and write file header. Set up restart/recovery.

process()

Select the new or changed orders in approved status for EDI download by fetching the driving cursor. Eligible orders are approved and have an EDI indicator set. The cursor selects “new” values from ordhead and “old” values from ordhead_rev for the next to last version (the last version is the current one, with the same information that is now on ordhead). For a new order, no earlier version will exist on the ordhead_rev table, so no “old” values will be fetched. If old values are null, this must mean that we have a **new** order. Information from different suppliers can be sent in the same file, but the file will be sorted by supplier.

Call write_TORDR to write order header level information to file. Call write_skus to fetch additional item-level information and write it to output file. Update the ordhead table to show that an EDI transaction has been sent and an acknowledgment has not yet been received.

write_TORDR()

Write the TORDR line (order header level information) to the output file.

write_skus()

Get SKU information (from the ordsku and ordsku_rev tables--quantity ordered, outstanding quantity, description). Get SKU cost information and supplier information (by calling get_supp_sku). If UPC information does not exist on ordsku, call get_upc to fetch the UPC information (if any). Call write_TSKUI to write SKU information line to file. If the SKU is a pack identifier (system_ind = 'P' on desc_look), call get_pack to get information on component items within the pack. If the order is to be pre-marked, call write_alloc to write allocation information to file; otherwise write shipment information to file by calling write_shipto.

write_TSKUI

Write SKU information line to file.

write_shipto()

Fetch shipment location and quantity information for a particular item on the order from the ordloc and ordloc_rev tables and call write_TSHIP to write it out to output file.

write_alloc()

This function is called only for cross-docked allocations that will be pre-marked by the supplier.

Fetch allocation information from the alloc_header, alloc_detail, and alloc_rev tables and call write_TSHIP to write it to output file.

get_pack()

Get information on skus contained within a pack (from the packsku table). If the item is part of a pack template, fetch the template description from the supps_pack_tmpl_desc table. Call get_supp_sku() to get the item_supplier for each of the skus. Use get_upc to fetch UPC information for these items. Call write_TPACK to write pack item information lines to file.

write_TPACK()

Write pack component information lines to file.

get_supp_sku()

Get supplier VPN, supplier's color code and supplier's size code from the item_supplier table.

get_UPC()

Get UPC info (either primary or preferred for supplier)

write_TSHIP()

Write TSHIP line to file—shipment location and quantity info. Also used to write allocation information

write_TTAIL()

Write order trailer line to file.

final()

Write file trailer, copy temporary file to final file (restart/recovery close), close files.

I/O Specification

For a new order, the “old” fields should be blank. For a changed order, both old and new fields should hold values, if value has changed. “Old” values come from the revision tables for the latest revision before the current one (the last one sent), while new orders come from the ordering tables.

FHEAD File identification. REQUIRED. One line per file.

TORDR Order level info REQUIRED. One line per order.

TSKUI item description REQUIRED. Multiple lines per order possible.

TPACK Pack contents. OPTIONAL. Multiple lines per order possible.

This line will be written only for pack items.

TSHIP Ship to location and quantity, allocation location REQUIRED. Multiple lines per item possible.

Allocation information is optional on this line—will exist if premark_ind is ‘Y’.

TTAIL Order end. REQUIRED. One line per order.

FTAIL End of file marker. REQUIRED. One line per file.

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record descriptor	Char(5)	FHEAD	File head marker
	Line id	Char(10)	0000000001	Unique line id
	Translator id	Char(5)	DLORD	Identifies transaction type
	File create date	Char(14)	Current date	YYYYMMDDHH24MISS format
TORDR	Record descriptor	Char(5)	TORDR	Order header info
	Line id	Char(10)		Unique file line id
	Transaction id	Char(10)		Unique transaction id
	Order change type	Char(2)		'CH' (changed) or 'NW' (new)
	Order number	Number(8)		Internal Retek order no
	Supplier	Number(10)		Internal Retek supplier id
	Vendor order id	Char(15)		External vendor_order_no (if available)
	Old order written date	Char(14)		Old date order created YYYYMMDDHH24MISS
	New order written date	Char(14)		Changed date order created YYYYMMDDHH24MISS
	Old Currency Code	Char(3)		Old order currency_code (ISO standard)
	New Currency Code	Char(3)		Changed order currency_code (ISO standard)
	Old Shipment Method of payment	Char(2)		Old ship_pay_method
	New Shipment Method of Payment	Char(2)		Changed ship_pay_method

Record Name	Field Name	Field Type	Default Value	Description
	Old Transportation Responsibility	Char(2)		Old fob_trans_res
	Old Trans. Resp. Description	Char(45)		Old fob_trans_res_desc
	New Transportation Responsibility	Char(2)		Changed fob_trans_res
	New Trans. Resp. Description	Char(45)		New fob_trans_res_desc
	Old Title Passage Location	Char(2)		Old fob_title_pass
	New Title Passage Location	Char(2)		Changed fob_title_pass
	Old Title Passage Description	Char(45)		Old fob_title_pass_desc
	New Title Passage Description	Char(45)		Changed fob_title_pass_desc
	Old not before date	Char(14)		Old not_before_date YYYYMMDDHH24MISS
	New not before date	Char(14)		Changed not_before_date YYYYMMDDHH24MISS
	Old not after date	Char(14)		Old not_after_date YYYYMMDDHH24MISS
	New not after date	Char(14)		Changed not_after_date YYYYMMDDHH24MISS
TSKUI	File record descriptor	Char(5)		SKU info
	Line id	Char(10)		Unique line id
	Transaction id	Char(10)		Unique transaction id

Record Name	Field Name	Field Type	Default Value	Description
	Old unit cost	Number(20)		Old unit_cost (4 implied decimal places)
	New unit cost	Number(20)		Changed unit_cost (4 implied decimal places)
	SKU	Number(8)		Sku (If a pack item, this will be the pack number)
	Old UPC	Char(13)		Old upc
	Old UPC supplement	Number(5)		Old upc_supplement
	New UPC	Char(13)		Changed UPC
	New UPC supplement	Number(5)		Changed upc_supplement
	Vendor catalog number	Char(30)		Supplier_sku (VPN)
	Free Form Description	Char(62)		Sku_desc
	Supplier Color	Char(80)		Supplier's color code (for fashion SKU's)
	Supplier Size	Char(80)		Supplier's size code (for fashion SKUS's)
	Pack Size	Number(4)		Supplier defined pack size
TPACK	File record descriptor	Char(5)	TPACK	Pack component info
	Line id	Char(10)		Unique line id
	Transaction id	Char(10)		Unique transaction id
	Pack id	Char(8)		Packsku.pack_no (same as sku for the pack item)
	Inner pack id	Number(8)		Inner pack identification
	Pack Quantity	Number(12)		Packsku.pack_sku_qty (4 implied decimal places)
	Component Pack Quantity	Number(12)		Packsku.comp_pack_qty (4 implied decimal places)
	Style Part Quantity	Number(12)		Packsku.style_pt_qty (4 implied decimal places)

Record Name	Field Name	Field Type	Default Value	Description
	SKU Quantity	Number(12)		Packsku.sku_qty (4 implied decimal places)
	SKU	Number(8)		Sku
	UPC	Char(13)		Upc
	UPC supplement	Number(5)		Upc_supplement
	VPN	Char(30)		Supplier sku (vpn)
	Supplier Color	Char(80)		Supplier's color code (for fashion SKU's)
	Supplier Size	Char(80)		Supplier's size code (for fashion SKU's)
	Style	Number(8)		Required when Pack Template is not NULL
	Pack template	Char(4)		Pack template associated w/style (packsku.pack_tmpl_id)
	Template description	Char(40)		Description of pack template (if present) sups_pack_tmpl_desc.supp_pack_desc
TSHIP	Record type	Char(5)	TSHIP	Describes file record-shipment info
	Line id	Char(10)		Unique file line number
	Transaction id	Char(10)		Unique transaction number
	Location type	Char(2)		'ST' store or 'WH' warehouse
	Ship to location	Number(4)		Location value form ordloc (store or wh)
	Old quantity	Number(12)		Old qty_ordered or qty_allocated (4 implied decimal places)
	New quantity	Number(12)		Changed qty_ordered or qty_allocated (4 implied decimal places)
	Old outstanding quantity	Number(12)		Old qty_ordered-qty_received (4 implied decimal places)(or qty_allocated-qty

Record Name	Field Name	Field Type	Default Value	Description
				transferred, for an allocation)
	New outstanding quantity	Number(12)		Changed qty_ordered-qty_received (4 implied decimal places)(or qty_allocated-qty_transferred, for an allocation)
	Cancel code	Char(1)		
	Old cancelled quantity	Number(12)		Previous quantity cancelled (4 implied decimal places)
	New cancelled quantity	Number(12)		Changed quantity cancelled (4 implied decimal places)
	Quantity type flag	Char(!)		'S'hip to 'A'llocate
	Store or warehouse indicator	Char(2)		'ST' (store) or 'WH' (warehouse)
	Old x-dock location	Number(4)		Alloc_detail location (store or wh)
	New x-dock location	Number(4)		Alloc_detail location (store or wh)
TTAIL	Record type	Char(5)	TTAIL	Describes file record – marks end of order
	Line id	Char(10)		Unique file line id
	Transaction id	Char(10)		Unique transaction id
	#lines in transaction	Number(10)		#lines in transaction
FTAIL	Record type	Char(5)	FTAIL	Describes file record – marks end of file
	Line id	Char(10)		Unique file line id
	#lines	Number(10)		Total number of transaction lines in file (not including FHEAD and FTAIL)

Test Conditions

Make sure to test for new orders, changed orders, and orders that contain pack items.

Sales and Stock Activity Report Download [edidlprd]

Design Overview

The following guideline describes Retek's implementation of the ANSI 852 Product Activity Data transaction set.

The productivity data report is a sales audit summary that will be sent to specified EDI vendors, giving sales details, current stock on hand for all location, and current in transit quantities for each of the SKUs primarily supplied by that vendor.

This program polls all suppliers that require activity reporting. Those that require daily reports will have daily sales information selected from the edi_daily_sales tables, while those requiring weekly reports will pull information from the WIN_STORE_HIST and RAG_SKUS_ST tables to get the weekly sales data. The store level tables WIN_STORE and RAG_SKUS_ST are queried for current stock levels, and the quantities that are in-transit are derived from the transfer tables (via a package call).

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
UNIT_OPTIONS	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No
WIN_STORE	No	Yes	No	No	No
WIN_STORE_HIST	No	Yes	No	No	No
RAG_SKUS_ST	No	Yes	No	No	No
RAG_SKUS_ST_HIST	No	Yes	No	No	No
RAG_STYLE	No	Yes	No	No	No
ITEM_SUPP_COUNTRY	No	Yes	No	No	No
ITEM_SUPPLIER	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE 4 (daily)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: Post program will delete old edi_daily_sales records

Threading Scheme: N/A (single threaded)

Restart Recovery

Will be threaded by supplier and the logical unit of work should be supplier.

Supplier Query

```
EXEC SQL SELECT TO_CHAR(s.supplier),
               s.sup_name,
               s.edi_sales_rpt_freq,
FROM sups s, v_restart_supplier rv
WHERE s.edi_sales_rpt_freq = 'D'
AND s.supplier = :ora_restart_driver_val
AND rv.num_thread = :ora_restart_num_thread
AND rv.thread_val = :ora_restart_thread_val
AND s.supplier > NVL(:ora_restart_supplier, -999)
ORDER BY s.supplier;
```

OR

```
EXEC SQL SELECT TO_CHAR(s.supplier),
               s.sup_name,
               s.edi_sales_rpt_freq,
FROM sups s, v_restart_supplier rv
WHERE s.edi_sales_rpt_freq in ('D', 'W')
AND s.supplier = :ora_restart_driver_val
AND rv.num_thread = :ora_restart_num_thread
AND rv.thread_val = :ora_restart_thread_val
AND s.supplier > NVL(:ora_restart_supplier, -999)
ORDER BY s.supplier;
```

Program Flow

N/A

Shared Modules

ITEMLOC_QTY_SQL.GET_TRANSFER_QTY –find quantity to transfer

Function Level Description

Init()

In addition to standard restart/recovery logic, this function should retrieve reporting date information. Use the cal_to_454_ldow function to determine the last day of the current week. The current vdate should be retrieved from period. If the vdate = current end of week variable then both daily and weekly reporting will occur, otherwise only daily processing will occur. The edi_sales_rpt_lag value should be retrieved from system_variables, further the current vdate – the lag days will be the reporting date for daily activity reports.

A count of all stores and warehouses should be done to determine the size of the item/location array and the output strings. The output strings (for on hand, in transit, and sales) will have to be dynamically sized to accommodate all location quantities. The string sizes should be total_locations (store + warehouse) * multiplier (macro defined as 20). The size_array() function should be called to size the program arrays.

Open output file and write file header record.

Process()

The driving cursor of this program will be based on supplier. All edi suppliers should be selected according to reporting frequency. If the current date is the end of the reporting week (current 454 day is 7), then both the weekly and the daily reporting suppliers should be selected, otherwise only the daily suppliers should be selected. Call write_tran_head and write_tran_tail to write transaction head and tail records. Call process_skus() function to process sku detail records.

Process_skus()

This function loops through the skus on sku_supplier that are primarily supplied by the current supplier. The process_report_type function is called to do updates for reporting suppliers (to stock on hand and in transit quantities on edi_daily_sales, for a daily report) and to write to the output file.

Process_report_type():

Writes the TITEM item detail line to the output file via the write_TITEM function. If reporting frequency is daily, calls update_daily_stock to do database updates and write_daily_items to write output information. If reporting frequency is weekly, calls write_weekly_stock and write_weekly_sales functions.

Update_daily_stock()

Stock information for the current sku is fetched from the item/location tables. The In-transit quantity is derived using the GET_TRANSIT_QTY function. The stock information is inserted into the edi_daily_sales table with the current vdate as the transaction date. If the record already exists on the table (could have been inserted during daily processing of posupld), then update the record.

Write_daily_items()

All the records on edi_daily_sales should be selected into arrays for the current sku and daily reporting date (vdate – lag days). Loop through the arrays; for each item location in the loop call the write_TQUTY function to write information for each activity type. (Summing to get total quantities will now be done by Gentran, not within this program).

Write_weekly_sales

Get weekly sales information from win_store or rag_skus_st and call write_TQUTY to write to output file.

Write_weekly_stock()

Get weekly stock information and call write_TQUTY to write to output file.

Write_tran_head()

Write transaction header to output file. Three dates are included: vdate-lagdays for a supplier that reports daily, and beginning or reporting week and end of reporting week for suppliers that report weekly.

Vendor sku will be written if exists for non-fashion item, otherwise, Retek sku number will be written. Fashion skus should have the vendor style written if it exists. If it doesn't exist then the Retek SKU should be written.

Write_TITEM:

Writes item number and description information to output file.

Fashion skus will have their colour and size appended to the style description.

Write_TQUTY

Write location, quantity, and quantity type (in-transit, sales, etc) information to output file.

Size_arrays()

Size the following arrays with a #define value: supplier array (from driving cursor) and sku_supplier array (skus for supplier w/primary_supplier_ind = Y). Sales and stock array (select sales information from edi_daily_sales/item_location_hist tables) should be dynamically sized according the number of stores and warehouses in system.

I/O Specification

Output file structure:

FHEAD file identification REQUIRED. One line per file.

THEAD Dates and supplier REQUIRED. One line per transaction.

TITEM Item information REQUIRED. Multiple lines per transaction possible.

TQUTY Location and quantity information REQUIRED. Multiple liens per transaction possible.

TTAIL End of transaction marker. REQUIRED. One line per transaction.

FTAIL End of file marker. REQUIRED. One line per file.

Record Name	Field Name	Field Type	Default Value	Description
File Header	File record descriptor	Char(5)	FHEAD	Describes record type
	Line number	Number(10)	0000000001	Sequential file line number
	File source	Char(5)	DLPRD	File source id for Gentran
	File create date	Char(8)	Period.vdate	Current date YYYYMMDD
Transaction header	File record descriptor	Char(5)	THEAD	Identifies record type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Report date	Char(8)		Vdate-lag days between reporting (vdate if supplier reports weekly) YYYYMMDD

Record Name	Field Name	Field Type	Default Value	Description
	supplier	Number(10)		Supplier
Item detail	File record descriptor	Char(5)	TITEM	Identifies file record type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	SKU	Number(8)		SKU
	UPC	Char(13)		UPC
	UPC supplement	Number(5)		UPC_supplement
	Vendor catalog number	Char(30)		Supplier sku
	Unit cost	Number(20)		Unit cost (4 implied decimal places) from sku_supplier table (in supplier currency)
	Item description	Char(62)		Item description (SKU desc)
Quantity detail	File record descriptor	Char(5)	TQUTY	Identifies record type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Quantity descriptor	Char(15)		'On-hand' (stock)/'Sold' (sales)/'In transit'
	Location type	Char(2)		'St'ore or 'WH' warehouse
	Location	Number(4)		Retek store or warehouse number
	Quantity	Number(12)		Quantity - 4 implied decimal places
Transaction trailer	File record descriptor	Char(5)	TTAIL	Identifies record type

Record Name	Field Name	Field Type	Default Value	Description
	Line number	Number(10)		Sequential file line number
	Transaction lines	Number(6)		Number of lines for this transaction
File trailer	File record descriptor	Char(5)		Identifies record type
	Line number	Number(10)		Total # lines in file
	Number of transaction lines	Number(10)		Number of transaction lines in file

Technical Issues

N/A

EDI Purge [ediprg]

Design Overview

This program will purge rejected EDI “new items” or cost changes on the edi temporary tables. New or changed item information as well as cost change information sent by suppliers will be approved or rejected on-line. The items that are rejected will be removed from the system when they exist for longer than the number of days specified on the system_variables table.

TABLE	SELECT	INSERT	UPDATE	DELETE
SYSTEM_OPTIONS	Yes	No	No	No
EDI_NEW_ITEM	No	No	No	Yes
EDI_COST_CHANGE	No	No	No	Yes

Scheduling Constraints

Processing Cycle: Ad Hoc

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

This program will use the commit_max_ctr on the restart_control table to periodically commit SQL delete operations.

Program Flow

N/A

Shared Modules

Restart_init() - initialize restart recovery tables

Restart_close() - commit changes and update restart recovery tables

Function Level Description

Init()

This function should call restart_init function to initialize the restart_recovery process. The g_l_restart_max_counter is assigned by the restart_init function from the max_commit_ctr on the restart_control table. This counter will be used in the process function to perform periodic commits. This function will also query the system_variables table to fetch the current values of edi_new_item_days and edi_cost_change_days, and will query the period table for the current vdate.

Process()

This function should delete rejected records from the edi temp tables that are older than the number of days specified by the edi_new_item_days and the edi_cost_change_days variables. A while loop is used to force periodic commits of the deletes based on the value of g_l_restart_max_counter. This is done to ensure that rollback segments will not be exceeded in the event of considerable volume

Final()

This function should call the restart close function to update the restart recovery tables with the final program status (completed if finished successfully, aborted if handled exceptions occur).

I/O Specification

N/A

Technical Issues

N/A

Supplier Order Acknowledgements and Changes [ediupack]

Design Overview

This program has four functions: to acknowledge vendor receipt of a buyer generated order without changes, to acknowledge vendor receipt of a buyer generated order with date, cost or quantity modifications, to notify buyer of a vendor-generated order, and to acknowledge order cancellations.

All acknowledgements will update the ordhead table with acknowledgement information.

When the supplier sends the acknowledgement with modifications, they can send the entire purchase order or only the changes. The file details will be matched to the current order. If the Not Before Date, Not After Date, Quantity, Price, and SKU all match the current order, then no changes were submitted. If one of the variables is blank, for example the price, assume that no pricing changes were made. As soon as one of the variables does not match, the order has been changed. These changes will not be written directly to the order; they will be written to the revision tables. Revisions will be accepted in the ordering dialog and changed orders will be resubmitted via edidlord. The revision number will be the incremented value of the last (or maximum) revision number for revisions of the order on the revision tables.

Vendor generated orders will create new orders by inserting new records on the EDI temporary order tables.

This program uses linked list processing. The linked list should hold one transaction at a time (THEAD, transaction header, through TTAIL, transaction tail). If there is an error with a transaction set, that transaction will be written to the reject file so it can be fixed and reprocessed later, then the program will continue to the next transaction set.

Vendor generated orders:

Item, location, supplier, and cost information are all required for this transaction type. Retek only supports single shipping locations for vendor replenishment. Once all required segments are read and validated, order information will be written to either the edi_ord_temp (staple items) or the edi_ord_tempf (fashion items) tables. The ordhead table will be updated to reflect that the PO acknowledgement was received.

Acknowledgement without change: Items that are read will be read and validated against the current order information. The ordhead table will be updated to reflect that the PO acknowledgement was received.

Acknowledgment with change:

The following fields can be modified on an existing buyer generated order: the not before date, not after date, item cost, qty ordered, and order location (for multiple shipping locations only). Date information that is sent will be compared to the dates stored on the ordhead table. If a difference is detected, a revision will be written to the ordhead_rev table for the order. Cost information that is sent will be compared to the cost stored on the ordloc tables. If a difference is detected then revisions will be written for the item and all of its shipping locations with the new cost information. Quantity information will be compared with the information on the ordloc table. If a difference is detected in the sent quantity and the stored quantities, then a record is inserted/updated on ordloc_rev with the new order quantity and the ordsku_rev record for that item is adjusted to reflect the difference in the quantity.

New items and shipping locations can also be sent. New item information is added to the ordsku_rev and the ordloc_rev tables. New locations for existing items are inserted to the ordloc_rev tables.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
EDI_ORD_TEMP	No	Yes	Yes	Yes	No
WIN_SKUS	Yes	Yes	No	No	No
SKU_SUPPLIER	Yes	Yes	No	No	No
UPC_EAN	Yes	Yes	No	No	No
ORDHEAD_REV	Yes	No	Yes	Yes	No
ORDSKU_REV	Yes	No	Yes	Yes	No
ORDLOC_REV	Yes	No	Yes	Yes	No

Scheduling Constraints

Processing Cycle:	Ad Hoc in Phases 1,2, or 3
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	Before Open to Buy processing
Threading Scheme:	N/A

Restart Recovery

The files will not have enough volume to warrant the implementation of restart recovery for commit/rollback considerations but minimal restart/recovery capability will be added.

Program Flow

N/A

Shared Modules

NEW_ITEM_LOC

NEW_PACK_LOC

PRICING_ATTRIB_SQL.BUILD_PACK_RETAIL

SUPP_ITEM_ATTRIB_SQL.GET_ATTRIB_FOR_ORDSKU

Function Level Description

Init:

Get vdate from period; open input file and reject file.

Process

This function reads and validates the transaction header. Then the item information lines and shipment location lines are read and processed, if present.

The ordhead table will be updated to reflect that a PO acknowledgement has been received after the transaction has been processed.

get_line:

Read a line from the input file and add it to the linked list.

dump_to_reject:

Read in the rest of a rejected transaction and write it to the reject file.

validate_THREAD

This function processes the THREAD line for purchase order acknowledgement. Variables on the line will be validated.

If the acknowledgement is a vendor generated order (vendor replenishment) set acknowledgement type and changed order flags. No more validation or processing will occur in this function for vendor generated orders.

Buyer order acknowledgement, with or without changes, retrieve the supplier number and the acknowledgement flag from ordhead. The edi acknowledgement must not have been received, and the supplier will be validated with the supplier value supplied in the file (if present). If the order is being changed (type AC) then the next revision number is retrieved from the ordloc_rev table for the order.

The supplier number read in this function should be validated against the supplier number retrieved from the tables given the order number. This validation will not occur for vendor generated orders because the supplier number is not fetched in that case.

The dates retrieved will be used for vendor generated orders or order date changes. Vendor generated orders and date changes require two dates, the not before date and the not after date.

validate_TITEM

This function processes the item data for purchase order acknowledgements. New or modified item quantities are determined, as well as item IDs and (SKU, VPN, ISBN, & UPC). This function will call the process_SKU function to get required SKU/UPC/VPN information regarding order items.

Also, this function processes the pricing information for purchase order acknowledgements. If the order is vendor generated then no further processing is required. If it is a buyer generated order then further validation and program logic may be required. If the item is new and it has a single shipping location then the add_new_item function is called to add ordsku_rev and ordloc_rev data (If the order has multiple shipping locations then the add_new_item() function will be called from the validate_TSHIP function.) New ordsku_rev and ordloc_rev records will be added for the item/location and other ordloc_rev records will be added via the add_new_loc function for further locations for the new item). If the item already exists, then the check_cost_chg function will be called to determine the sent cost information differs from the current order cost information. If this is the case the update_rev_cost function will be called from the check_cost_chg function.

validate_TSHIP

This function processes the destination quantity line for purchase orders. After location and quantity location information is read, the add_new_item function will be called if the item is new to the order. Otherwise the check_qty_chg function is called to see if the sent item/location quantity differs from the item/location quantity currently held on ordsku for the order. If the quantities differ the update_rev_qty function will be called from the check_qty_chg function to update insert new information on the ordsku_rev and ordloc_rev tables.

cancel_order:

This function inserts into the ordhead_rev, ordloc_rev, and ordsku_rev tables. A zero quantity is written for the sku to be cancelled.

process_sku()

This function is called after the item information is read from the file. UPC items should retrieve the Retek SKU number from upc_ean and VPN items (Vendor Catalog) should retrieve the Retek SKU number from sku_supplier for the VPN and supplier values in the EDI file. Non-UPC items, SKU, VPN, & ISBN numbers, will retrieve the UPC and UPC supplement from the upc_ean table. Buyer generated orders will check to see if current item is on the ordsku table for the order. If the item does not exist, it is flagged as a new item.

Check_dates()

This function will be called by validate_THEAD. The dates read from the input file will be compared with the dates on the ordhead table. If either date has changed, then call the update_rev_dates() function to add a new revision with the modified not_before_date and not_after_date to the ordhead_rev table.

Check_cost_chg()

This function will be called by validate_TITEM for existing items. The edi cost value read from the input file will be compared with order costs on the ordloc table. If any item/location has a different cost than the passed cost segment element then call the update_rev_cost() to adjust the cost amounts on the ordloc_rev records and the cost_source field on the ordsku_rev tables.

Check_qty_chg()

This function will be called the validate_TSHIP function. The sent quantity will be compared with the quantity stored on the ordloc table for the item/location. If the quantities differ or if the record does not exist (meaning the location does not exist for the order) then the update_rev_qty function will be called to update/insert the proper quantities on the ordloc_rev and the ordsku_rev tables.

Update_rev_dates()

This function will be called from the check_dates() function when passed a not_before_date and/or not_after_date that differ from the dates on the ordhead table. The function should insert a new record into ordhead_rev with the revised dates (if only one date has been revised, get the original value of the other date off the ordhead table and insert that value instead).

Update_rev_cost()

This function will be called from the check_cost_chg() function when a passed cost amount differs from any item/location cost amount on the ordloc table. The function should update the ordsku_rev cost_source field to 'MANL' for the current order, sku, and revision number. Updates/Inserts should occur in the following manner:

- If the record exists then the ordloc_rev unit_cost field should be updated with the new unit cost for the order, sku, revision number and all locations. This scenario will only occur when a single item location has already processed a changed quantity (therefore all ordloc records will be accounted for, as there will only be one).
- If the record does not exist on the ordloc_rev table, then a record should be inserted into the ordsku_rev table with the new revision number and cost_source of 'MANL'. All other information should be selected from ordsku for the current order and item. Then a record should be inserted into the ordloc_rev table for every record on ordloc. The new cost should be used, as well as the new revision number, but all other information should be selected from ordloc.

Update_rev_qty

This function will be called from the check_qty_chg() function when a passed qty amount for an item/location differs from the qty amount on the ordsku table for that order, item, & location. The function should update the revisions tables in the following manner:

- If the item is not a new location (item-location not on ordloc table - determined in check_qty_chg function), update the ordered quantity on ordloc_rev for the item, location, and revision number. If the record exists then update the ordsku_rev ordered quantity with the difference between the existing order quantity on ordloc and the new ordered quantity from the EDI input file. Also, set a flag to update the ordsku_rev table with the adjusted order quantity.
- If the ordloc_rev quantity is unsuccessful or if the location is new, then do the following:

Try to insert a record onto the ordsku_rev table. The new record values should include the revision number, the values from the ordsku table for the order and item, and the adjusted order quantity. Error checking should accommodate primary key constraint violations. New locations may or may not have an ordsku_rev record for the current revision. If the ordsku_rev record already exists then set flag to update the existing record with the adjusted order quantity.

If there is a new location for the item (doesn't exist on the ordloc table), call the add_new_location function to add the new ordloc_rev record information.

If the location already exists on the ordloc table, then insert a new record onto ordloc_rev based on the values on ordloc for the order, item, and location; but with the new revision number and the new ordered quantity.

- If the update ordsku_rev flag is set to true (see cases above), then update the ordsku_rev record for the current order, revision number, and sku. Subtract the existing ordloc ordered quantity from record's ordered quantity and add to it the new order quantity read from the EDI input file.

Update_replenish()

This function will be called to add records to the vendor replenishment tables. If the order is a vendor generated then new records will be inserted onto edi_ord_temp (or updated on this table if present). Information should be derived from the input file and the item tables.

Add_new_item()

This function will insert new records on the ordsku_rev table for a SKU new to the existing Retek buyer order. Insert new information to the ordsku_rev table with the following values: new revision number, 'R' (revision type), edi order number, blank case id, edi sku, edi upc, edi upc supplement, sku description from win_skus, null colour description, order_qty from input file, 0 for qty_received, cost source is 'MANL' if processed CTP segment for item, otherwise it's 'NRML', edi code type is null, prepack indicator is null, assortment indicator is null, and the rev_date is the vdate.

The add_new_loc function should be called to insert new ordloc_rev records.

Add_new_loc()

The function will insert records on the ordloc_rev tables for new locations on an order. It will be called by the add_new_item function and from the update_rev_qty function. Determine whether the new location is a store or a warehouse. New information should be added from the edi order. Unit retail should be derived from win_skus for the item. The cost will be the passed cost if the cost was in the file for the item, otherwise it should be take from the win-skus table.

Get_location()

This function will assign values to store and warehouse variables depending on the value of the location value read from the edi input file. The store and warehouse tables should be hit to verify the location's existence.

Get_item_info

This function will retrieve sku information for inserts to edi_ord_temp and the revisions tables. It is called from add_new_item, add_new_loc. System_ind, dept, and sku description will be retrieved from the desc_look table. Cost and retail are retrieved from the appropriate item tables (based on the system indicator).

get_item_info_wrap:

Call get_item_info and also make_new_loc if necessary.

make_new_loc:

Call the NEW_ITEM_LOC procedure to create a new sku/location relationship

make_new_pack_loc

Call the NEW_PACK_LOC procedure to create a new pack/location relationship

get_pack_retail

Call PRICING_ATTRIB_SQL.BUILD_PACK_RETAIL to get the unit retail price for a pack

free_record_memory

Wipe clean the linked list containing all records on a transaction (stored in case the transaction needs to be rejected)

add_detail_record_to_list:

Add a record to the linked list

get_pack_ind

Get the simple_pack_ind from packhead for a given pack.

final:

Restart/recovery close; close input and reject files.

I/O Specification

Input file:

FHEAD file identification REQUIRED—one line per file

THEAD order and acknowledgment info REQUIRED—One line per acknowledgment

TITEM Item information OPTIONAL---multiple lines per acknowledgement possible

TSHIP location and quantity information OPTIONAL—multiple lines per item possible

TTAIL acknowledgment end marker REQUIRED—one line per acknowledgement

FTAIL file end marker REQUIRED—one line per file

Record Name	Field Name	Field Type	Default value	Description
FHEAD	Record descriptor	Char(5)	FHEAD	Describes file line type
	Line number	Number(10)	0000000001	Sequential file line number
	Retek file ID	Char(5)	ORDAC	Describes file type
THEAD	Record descriptor	Char(5)	THEAD	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Acknowledge type	Char(2)		AP-product replenishment AK-no detail or change (use for UK receive, also) (NEW:CA-cancel order (no detail))

Record Name	Field Name	Field Type	Default value	Description
	Order number	Char(15)		May be external order number (vendor order number) OR Retek order number
	Written_date	Char(8)		Written date YYYYMMDD format
	Supplier number	Number(10)		Retek supplier number
	Not before date	Char(8)		Not_before_date YYYYMMDD
	Not after date	Char(8)		Not_after_date YYYYMMDD
TITEM	Record descriptor	Char(5)	TITEM	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	SKU	Number(8)		SKU **At least one of SKU, UPC
	UPC	Char(13)		UPC **vendor item# must exist
	UPC supplement	Number(5)		UPC supplement
	Vendor catalog number	Char(30)		Vendor item number (supplier_sku)
	Unit cost value	Number(20)		Unit_cost (4 implied decimal places)
TSHIP	Record descriptor	Char(5)	TSHIP	Describes file line type
	Line number	Number(10)		Sequential file line number

Record Name	Field Name	Field Type	Default value	Description
	Transaction number	Number(10)		Sequential transaction number
	Store/wh indicator	Char(2)		'ST' for store, 'WH' for warehouse
	Ship to location	Number(4)		Store or warehouse number
	Quantity	Char(12)		Quantity ordered (4 implied decimal places)
TTAIL	Record descriptor	Char(5)	TTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Lines in transaction	Number(6)		Total number lines in this transaction
FTAIL	Record descriptor	Char(5)	FTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number(total # lines in file)
	Number of transactions	Number(10)		Number

Technical Issues

N/A

New and Changed Supplier Address Upload [ediupadd]

Design Overview

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

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

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

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

Scheduling Constraints

Processing Cycle: Ad Hoc

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

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

Restart Recovery

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

Program Flow

N/A

Shared Modules

N/A

Function Level Description

Init()

Open input & output files. Validate file header line. Call Retek API restart_init to initialize restart/recovery process.

Process()

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

Update_supplier()

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

validate_FDETL—validate supplier and address information

I/O Specification

Input file:

FHEAD File type identification

FDETL Supplier address info

FTAIL End of file marker

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	File record descriptor	Char(5)	FHEAD	Describes file line type
	Line number	Number(10)		Sequential file line number
	Gentran_id	Char(5)		Identifies the file type
	File create date	Char(14)		YYYYMMDDHH24MISS format

Record Name	Field Name	Field Type	Default Value	Description
FDETL	File record descriptor	Char(5)	FDETL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Add or Update	Char(1)		‘A’dd or ‘U’pdate address
	Address type	Char(2)		Will be translated into sups_add.address_type: 01 – Business 02 – Postal 03 – returns 04 – Pick Up (Order) 05 – Payment
	Supplier	Number(10)		Sups.supplier
	Address line 1	Char(30)		Sups_add.address_1
	Address line 2	Char(30)		Sups_add.address_2
	Address line 3	Char(30)		Sups_add.address_3
	Contact name	Char(20)		Sups_add.contact_name
	Contact Phone	Char(20)		Sups_add.contact_phone
	Contact fax	Char(20)		Sups_add.contact_fax
	City	Char(20)		Sups_add.city
	State	Char(3)		Sups_add.state
	Postal code	Char(10)		Sups_add.post_code
	Country	Char(3)		Sups_add.country_id
FTAIL	File record descriptor	Char(5)		Describes file record type
	Line number	Number(10)		Sequential file line number (total # lines in file)
	Number of transactions	Number(10)		Number of transactions in file

Technical Issues

N/A

ASN Upload [ediupasn]

Design Overview

The following guideline describes Retek's implementation of the Shipment Manifest Transaction Set. Retek uses this document to receive advance notice of shipments and what orders and items are contained within those shipments.

A new SHIPMENT record is inserted for every new shipment block consisting of orders and individual locations of the order. A separate sku record is inserted into the SHIPSKU table for all the item blocks. The shipment is created with a STATUS of 0 or 6 that indicates the shipment record is an ASN or ASN carton.

However, if receiving is processed prior to the arrival of ASN (i.e. goods arrive before ASN), a receipt shipment with the same order number/ASN number combination has already been created in the receiving module (rcvupld.pc) with ship_origin=4 and status_code=U. In this case, this receipt shipment will be updated and a new shipment will not be created. In addition, a new invoice is inserted for the receipt shipment. Items that exist on the ASN but do not exist on receipt shipment will be inserted into the shipsku table with qty_received = 0 and will not be included on the invoice.

Retail Functionality:

Based on Standard Carton Packs -> identical SKUs in each carton.

Each shipment is based on a single Ship From / Ship To relationship.

Shipments will be set up with the 'shipsku' records based on cartons for all records of the pick and pack structure. The carton information is a required level for this structure to process. All 'shipsku' records will be created with a status of accept since there is no distinct number to compare against the ordloc table.

The input file will be in a standard Retek format, instead of the EDI format previously used. The Gentran file translator will handle the translation from EDI to the Retek format from now on.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
SYSTEM_VARIABLES	Yes	Yes	No	No	No
SHIPMENT	No	No	Yes	No	No
SHIPSKU	No	No	Yes	Yes	No
ORDHEAD	Yes	Yes	No	No	No
ORDSKU	Yes	Yes	No	No	No
ORDLOC	Yes	Yes	No	No	No
UPC_EAN	No	Yes	No	No	No
ITEM_SUPPLIER	No	Yes	No	No	No
ITEM_SUPP_COUNTRY	No	Yes	No	No	No
CARTON		No	Yes	No	No
INVC_HEAD	Yes	No	Yes	Yes	No
INVC_XREF	Yes	No	Yes	No	No
INVC_DETAIL	Yes	No	Yes	Yes	No

Scheduling Constraints

Processing Cycle: PHASE 1 (daily)

Scheduling Diagram: May be run multiple times during the day, but should be completed before processing any receiving uploads from store or warehouse systems.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Driven by the number of distinct ASN files to be processed.

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

NEXT_SHIPMENT—get new shipment number

INVC_WRITE_SQL.ASN_TO_INVC – write new invoice for received items.

Function Level Description

Init:

Open input and error files. Do restart initialization.

Process:

Read in each transaction and process. Validation occurs on shipment, carton, order, and item level; table inserts and updates occur on item level.

validate_TSHIP:

Parse the TSHIP line into variables and validate. Make sure that the to location exists and determine if it's a store or a warehouse. Check that the order exists. Check what the pre_mark_ind on ordhead is for this order number.

validate_TORDR:

Parse the TORDR line into variables and validate. Check the not-after date against the not after date from ordhead. If no_boxes was not given a value on the THEAD line, calculate it here from the lading_qty.

validate_TCART:

Parse the TCART line into variables and validate. If pre_mark_ind is 'Y', insert records to the carton table for each allocation.

validate_TITEM:

Parse the TITEM line into variables and validate. Determine what item type (SKU, UPC, vendor catalog number.) was given. If traded_units has a value, use it to calculate the quantity shipped.

Call check_item and check_quantity.

check_item:

Make sure item exists and is on order and get its sku number if that was not in the input file. If a shipment already existed (before running the program), call match_item. Otherwise insert the item into the shipsku table.

match_item:

Select the quantity received from shipment and shipsku for the item. If no data was found, insert item into shipsku.

check_quantity:

Select quantity ordered, unit cost, and unit retail from ordloc where the order number, sku, store, and warehouse match. If this is a new shipment (inserted by this program): do either 1)if the quantity shipped was greater than the quantity ordered and we are not doing carton receiving, set the status_code to H (hold) on shipsku. or 2)If carton receiving is being done, set status_code to A. Always update shipsku with the quantity being shipped, unit cost, and unit retail. If the shipment existed before: 1)if the sku didn't exist before, update shipsku qty_expected, unit cost, and unit retail. 2) If the sku did exist, update status_code and qty_expected.

match_shipment:

Check shipment table for an existing shipment with the current vendor order number, store, and warehouse. If there is a match, insert a record into the AP_tran table and update the status code to R (received) on all matching shipment table records.

new_shipment:

Check ordloc table for existence of the order number, store, and warehouse. (non-fatal error if don't exist). Use next_shipment to get a new shipment number. Insert into shipment table.

Final:

Close files and call restart close.

I/O Specification

Input file structure:

TCART lines will only be present if the order contains cartons. All other lines are required.

FHEAD

TSHIP shipment information (One line per shipment)

TORDR order information (Can be multiple lines per shipment)

TCART carton information (Optional)

TITEM item information (Can be multiple per order)

CTAIL (if TCART line exists)—end of carton marker

OTAIL end of order marker

TTAIL end of transaction (shipment) marker

FTAIL

All character variables should be right-padded with blanks and left justified; all numerical variables should be left-padded with zeroes and right-justified. Missing variables should be blank.

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	File head descriptor	Char(5)	FHEAD	Describes file line type
	Line id	Char(10)	0000000001	Sequential file line number
	Gentran ID	Char(5)	UPASN	Identifies which translation Gentran uses
	Current date	Char(14)		Vdate in YYYYMMDDHH24MISS format
TSHIP	File record descriptor	Char(5)	TSHIP	Shipment information
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number

Record Name	Field Name	Field Type	Default Value	Description
	Vendor shipment #	Char(15)		Shipment.ext_shipment-external shipment #
	Shipment date	Char(14)		Shipment.ship_date YYYYMMDDHH24MISS
	Scheduled delivery date	Char(14)		YYYYMMDDHH24MISS
	Scheduled shipment date	Char(14)		YYYYMMDDHH24MISS
	Courier	Char(20)		Shipment.courier
	Supplier	Number(10)		Supplier
	Destination location	Number(4)		Warehouse or store being ordered to on the ordloc table.
	Payment method	Char(2)		Ordhead.ship_pay_method
TORDR	File record descriptor	Char(5)	TORDR	Describes file record type
	Line ID	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Order number	Number(15)		Retek order number or vendor order number
	Packaging code	Char(3)		Describes carton type
	Lading quantitiy	Number(4)		Used to calculated no_boxes
	Not after date	Char(14)		Ordhead.no_after_date YYYYMMDDHH24MISS
	Carton indicator	Char(1)		'Y' if cartons present, 'N' if not

Packaging code to be populated from the following table: (example:CTN). CTN is the only value used by the system; others are ignored.

BAG	Bag
CTN	Carton
PLT	Pallet

Record Name	Field Name	Field Type	Default Value	Description
TCART	File record descriptor	Char(5)	TCART	Describes file record type
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	UCC-128 Carton number	Char(20)		Skipsku.carton
	Allocation location	Number(4)		Allocation location (blank if none)
TITEM	File record descriptor	Char(5)	TITEM	Describes file record type
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	SKU	Number(8)		Sku
	UPC	Char(13)		Upc
	UPC-supplement	Number(5)		Upc_supplement
	Vendor product number	Char(30)		Supplier sku
	Quantity shipped	Number(12)		Quantity shipped (4 implied decimal places)
	Pack size	Number(4)		Supplied pack size (blank if not shipped in pack)
CTAIL	File record descriptor	Char(5)	CTAIL	Marks end of carton

Record Name	Field Name	Field Type	Default Value	Description
	Line id	Char(10)		Sequential file line number
	Carton lines	Number(6)		Number of lines between TCART and CTAIL
OTAIL	File record descriptor	Char(5)	OTAIL	Marks end of order
	Line id	Char(10)		Sequential file line number
	Order lines	Number(6)		Number of lines between TORDR and OTAIL
TTAIL	File record descriptor	Char(5)	TTAIL	Describes file record type
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Transaction lines	Number(6)		Total number of lines in transaction
FTAIL	File record descriptor	Char(5)	FTAIL	Describes file record type
	Line id	Char(10)		Sequential file line number
	File transaction lines	Number(10)		Total number of lines less file head and tail lines

Technical Issues

N/A

Supplier Availability for Contracts Upload [ediupavl]

Design Overview

This program uploads supplier availability.

Scheduling Constraints

Processing Cycle:	Daily, Phase 1
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A –file based processing

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

EDIUK_SQL.SUPPLIER_EXISTS—check for existence of item on sku_supplier table

EDIUK_SQL.SUP_AVAIL_INSERTS—do inserts and updates to the supp_avail table

Function Level Description

Init:

Open files

Call restart_file_init

File_process:

Loop:

 read from input file into structure array

 validate header and detail fields with validate_record

 process records with process_record

 call restart_file_write and restart_commit

 end loop

Validate_record:

Parse record into variables and validate

Check that item exists on sku-supplier tables by calling EDIUK_SQL.
SUPPLIER_EXISTS

Process_record:

Call EDIUK_SQL.SUP_AVAIL_INSERTS to do inserts and updates to
supp_avail table

Final:

Call restart_file_write

Close files

Remove temp file

I/O Specification

Upload file format:

FHEAD—file identification

FDETL—supplier, item, available quantity

FTAIL—total number of detail records sent

All character variables should be right-padded with blanks and left_justified; all numerical variables should be left-padded with zeroes and right-justified. All dates should be in YYYYMMDD24MISS format.

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record descriptor	Char(5)	FHEAD	Describes file line type
	Line number	Number(10)	0000000001	Sequential file line number
	File type	Char(4)	SPAV	
	Create date	Char(14)		File create date YYYYMMDDHH24M ISS format
FDETL	Record descriptor	Char(5)	FDETL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction number	Number(14)		Sequential transaction number
	Supplier	Char(10)		
	Item type	Char(3)		'SKU', 'UPC', or 'VPN' item type
	Item id	Char(30)		Actual SKU/UPC/VPN number
	Item supplement	Char(5)		UPC supplement
	Available quantity	Char(12)		Available quantity*10000 (4 implied decimal places)
FTAIL	Record descriptor	Char(5)	FTAIL	Describes file line type
	Line	Number(10)		Sequential file line

Record Name	Field Name	Field Type	Default Value	Description
	number			number (total # lines in file)
	Number of detail records	Number(10)		Number of FDETL lines in file

Technical Issues

N/A

New and Changed Upload from Supplier [ediupcat]

Design Overview

The purpose of the ediupcat batch program is to update the edi_new_item and edi_cost_change tables. This will allow the users to view and implement the vendor changes online instead of manually viewing and inserting information.

EDIUPCAT will read in a file and strip out the appropriate information. For each line item, the supplier has the option of sending one or all of the following: vendor style, SKU, UPC, and VPN numbers. If a SKU number is sent, this implies that the item exists in Retek. This number is validated against the SKU tables – UPC and VPN are validated if present. If the SKU number is not present in the file, the program searches Retek for the SKU number. A new vendor style may be sent by sending vendor style id and style information, without a specific item id. The item or style information is then updated or inserted into the edi_new_item and edi_cost_change tables based on the data processed.

It is permissible for new items to be sent with new vendor styles, but only if the new vendor style is also present in the file (this constraint is for the purposes of creating a Retek style id in the editem form , which will then be applied to all items with the associated vendor style)

All input file information is validated. Any erroneous data will cause the entire transaction to be written to a run-time rejection file.that can be reprocessed once the appropriate adjustments are made.

The batch program will have the ability to process multiple transactions per file.

The input file format will be in a Retek standard file format, rather than EDI format. The translation from EDI to this standrd format will be done by the Gentran translator.

Scheduling Constraints

Processing Cycle:	Daily, Phase 2
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	(File-based processing, don't use multithreading)

Restart Recovery

The EDIUP832 batch program will use restart/recovery initialization, close, and intermittent commits (restart_commit)

Program Flow

N/A

Shared Modules

ITEM_VALIDATE_SQL.EXIST –check if item already exists in system

ITEM_VALIDATE_SQL.SKU_UPC—check that UPC matches SKU

SQL_LIB.BATCH_MSG—to write error messages

FASHION_ATTRIB_SQL.GET_STYLE—get style given SKU

ITEM_ATTRIB_SQL.TYPE—get item type

CURRENCY_SQL.CONVERT_BY_LOCATION—convert unit cost and unit retail

COUNTRY_VALIDATE_SQL.EXISTS_ON_TABLE—validates origin_country_id

SYSTEM_OPTIONS_SQL.GET_ALL_DEFAULTS—retrieves default standard_uom, dimension_uom, weight_uom and packing_method.

UOM_SQL.GET_CLASS—retrieves the class that the UOM exists in

Function Level Description

init()

- get vdate
- Restart/recovery initialization
- open input file and read file header
- open output file (run-time reject file)

process()

- Read transaction header Loop:

Read transaction detail

Validate variables

If item, call process_sku:

If new item or change to existing item, insert into edi_new_item

If cost change, update edi_cost_change

If new style, call process_style_only:

insert into edi_new_item

validate_thead()

- parse thead line into variables and validate

validate_fdetl

- parse fdetl line into variables and validate

validate_SKU()

- If SKU on file, call item_validate_sql.exist to validate the SKU – if SKU does not exist, non-fatal error
- Call item_attrib_sql.type to retrieve the item type
- Ensure that the item type is either SKU, Fashion SKU, or Display Pack – if not, non-fatal error
- If the item is a Fashion SKU, call fashion_attrib_sql.get_style to retrieve the associated style – if no style, non-fatal error

- If UPC on file, call `item_validate_sql.sku_upc` – if UPC does not exist, non-fatal error
- if VPN on file, validate the VPN – if no VPN, non-fatal error

`validate_origin_country_id()`

New function:

If origin country id on file, call `country_validate_sql.exists_on_table` to validate the origin country. If origin country does not exist, write to error file.

`validate_UOMs()`

New function:

- If standard UOM on file, call `uom_sql.get_class` to get the UOM class. If the UOM class does not exist or the UOM class is in either MISC or PACK class, write to error file.
- If dimension UOM on file, call `uom_sql.get_class` to get the UOM class. If the UOM class does not exist or the UOM class is not in the DIMENSION class, write to error file.
- If weight UOM on file, call `uom_sql.get_class` to get the UOM class. If the UOM class does not exist or the UOM class is not in the MASS class, write to error file.
- If any of the UOMs are not on file (or the packing method) call `SYSTEM_OPTIONS_SQL.GET_ALL_DEFAULT_UOM`. If the `standard_uom` or `packing_method` is NULL then directly fill the appropriate variable. However, populate the `dimension_uom` only if the dimension fields (length, width, height) are not NULL and populate the `weight_uom` only if the `ship_carton_wt` is not NULL.

get_sku_no()

- If item is a fashion SKU, we must have the UPC – if not, non-fatal error
- If UPC on file, get SKU from upc_ean table
- If the SKU is not found and VPN is on file, validate the VPN on the item_supplier table
- If the SKU number was retrieved
 - Call item_attrb_sql.type to retrieve the item type
 - Ensure that the item type is either SKU, Fashion SKU, or Display Pack – if not, non-fatal error
 - If the item is a Fashion SKU, call fashion_attrb_sql.get_style to retrieve the associated style – if no style, non-fatal error
- If the SKU number was not retrieved, check the edi_new_item table
- If the SKU number was not retrieved, it is a new SKU

process_sku()

- Check the edi_new_item table for the existence of SKU/supplier/origin_country combo
- Convert currency into primary currency for edi_new_item table – call currency_sql.convert_by_location
- If item is not on edi_new_item table
- If SKU exists
 - Update/insert edi_cost_chg table
- Insert into edi_new_item table – do not insert if item is a display pack ** insert cost and retail for new items only ** insert style for existing fashion SKUS
- If item is on edi_new_item table
 - Update edi_new_item table – do not update if item is a display pack
 - If SKU exists

- Update/insert edi_cost_chg table

vdate – processed date

NVL(sku_desc, edi_new_item.sku_desc)

NVL(unit_retail, edi_new_item.unit_retail) – for new items only

NVL(unit_cost, edi_new_item.unit_cost) – for new items only

NVL(length, edi_new_item.length)

NVL(height, edi_new_item.height)

NVL(width, edi_new_item.width)

NVL(weight, edi_new_item.weight)

NVL(lead_time, edi_new_item.lead_time)

NVL(min_ord_qty, edi_new_item.min_ord_qty)

NVL(max_ord_qty, edi_new_item.max_ord_qty)

NVL(supp_size, edi_new_item.supp_size)

NVL(supp_color, edi_new_item.supp_color)

process_style_only()

- insert style information into edi_new_item table

initialize_new_sku()

- initialize all oracle bind variables

reject_records()

- write transaction to reject file

- rollback to savepoint

final()

- restart/recovery close, close files

I/O Specification

Input file structure: (reject file will have same file structure)

FHEAD file header

FDETL item info

FTAIL file trailer

Record Name	Field Name	Field Type	Default Value	Description
File header	File record descriptor	Char(5)	FHEAD	Identifies file record type
	Line sequence	Number(10)	0000000001	Sequential file line #
	File type definition	Char(5)	UPCAT	Identifies program to use
	File creation date	Char(14)		YYYYMMDDHH24MISS format
Transaction detail	File record descriptor	Char(5)	FDETL	Identifies file record type
	Line sequence	Number(10)		Sequential file line #
	Transaction sequence	Number(10)		Sequential transaction #
	Supplier	Number(10)		Supplier id#
	Vendor style	Char(30)		Vendor style id
	Style	Char(8)		Retek style
	SKU	Number(8)		SKU (blank if none)
	UPC	Char(13)		UPC code (or ISBN) blank if none
	UPC supplement	Number(5)		UPC supplement (zeroes if none)
	UPC type	Char(5)		UPC type (blank if none)
	VPN	Char(30)		Vendor product number (blank if none) Must be in all capitals
	Color	Char(80)		Item color description

Record Name	Field Name	Field Type	Default Value	Description
	Size	Char(80)		Item size description
	Description	Char(40)		Item description
	Effective date	Char(14)		Effective date, YYYYMMDDHH24MISS
	Min order qty	Number(12)		Minimum order quantity (4 implied decimal places)
	Max order qty	Number(12)		Maximum order quantity (4 implied decimal places)
	Weight	Number(8)		Ship carton weight
	Length	Number(5)		Ship carton length
	Width	Number(5)		Ship carton width
	Height	Number(5)		Ship carton height
	Lead time	Number(4)		Days from PO receipt to shipment
	Unit cost	Number(20)		Unit cost, 4 implied decimal places
	Unit retail	Number(20)		Unit retail, 4 implied decimal places
	Pallet Size	Number(13)		Number of 'Eaches' that makes up a pallet
	Pack size	Number(13)		Supplied pack size
	Inner pack size	Number(13)		Supplied inner pack size
	Origin Country ID	Char(3)		Supplied origin country ID
	Standard UOM	Char(4)		Unit of measure for SOH
	UOM Conversion Factor	Number(20)		Conversion Factor, 10 implied decimal places
	Dimension UOM	Char(4)		Unit of measure of supplier pack dimensions
	Weight UOM	Char(4)		Unit of measure of supplier pack weight

Record Name	Field Name	Field Type	Default Value	Description
	Packing Method	Char(6)		Packing Method code (HANG,FLAT)
File tail	File record descriptor	Char(5)	FTAIL	Identifies file record type
	Line sequence	Number(10)		Sequential file line # (which is total # lines in file)
	Number of transactions	Number(10)		Number of transactions in file

ediupinv.pc

Design Overview

This program will upload invoice and credit memo information from an ASCII file (which will be translated into Retek standard format by the client from an EDI file) into the invoice matching tables.

Scheduling Constraints

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
SYSTEM_OPTIONS	Yes	Yes	No	No	No
PERIOD	Yes	Yes	No	No	No
ORDHEAD	Yes	Yes	No	No	No
UPC_EAN	No	Yes	No	No	No
INVC_HEAD		Yes	Yes	No	No
INVC_DETAIL		No	Yes	No	No
INVC_XREF		No	Yes	No	No
INVC_MATCH_QUEUE		No	Yes	No	No
INVC_MATCH_WKSHT		No	Yes	No	No
INVC_NON_MERCH		No	Yes	No	No
INVC_MERCH_VAT		No	Yes	No	No
TERMS		Yes	No	No	No
VAT_CODES		Yes	No	No	No
RTV_HEAD		Yes	No	No	No

Processing Cycle: Phase I (daily)

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A –file based.

Restart Recovery

The logical unit of work is the invoice.

Commits should occur every 2500 records (subject to experimentation).

Shared Modules

INVC_SQL.NEXT_INVC_ID—get a new invoice id

INVC_SQL.APPLY_XREF—given an order number, insert that order and received shipments associated with the order into invc_xref

Function Level Description

Set up structures for each different type of file line.

Input and reject file names should be accepted from the command line at runtime.

init()

Restart/recovery initialization

Fetch vdate from period and vat_ind and invc_match_qty_ind from system_options. Vdate will be used as the create date.

Open input file and reject file

Read in FHEAD line (or seek to the right place in the file, if doing a restart)

process()

Read in transactions from file (use get_line). As lines are read in, save to a linked list for rejection if necessary. For each transaction:

1. Deal with header-level information (THEAD line)—format the line (format_THEAD), do basic validation (validate_THEAD), get a new invoice number (get_new_invc_id) and insert into invc_head, invc_match_queue, and invc_non_merch tables. If order numbers are given, call insert_order_xref. If shipment numbers are given, call insert_ship_xref.
2. Deal with VAT information (TVATS lines—format_TVATS, validate_TVATS). If VAT is turned on in the system, call insert_merch_vat. (if turned off, read through the lines but don't process).
3. Deal with item_level information (TDETL line). Format, validate, insert into invc_detail table.
4. Deal with shipment/carton information (TSHIP line). Format, validate, insert into invc_match_wksht.
5. When get to TTAIL line, check number of lines in transaction from file against number of lines read in by program. If they don't match, reject the transaction.

6. If a transaction failed validation anywhere, roll back to a savepoint set at the beginning of the transaction and write the transaction to the reject file (write_to_reject)

7. When transaction has been processed, call restart commit logic if necessary.

get_new_inv_id()

Call the invc_sql.next_inv_id package to get a new invoice number

process_THEAD()

Format and validate header variables and do necessary table inserts.

process_TDETL()

Format and validate item-level variables and do necessary table inserts.

process_TVATS()

Format and validate TVATS variables and insert into invc_merch_vat.

process_TSHIP()

Format and validate TSHIP variables and insert into invc_match_wksht.

process_FTAIL()

Validate FTAIL line from input file and create new FTAIL line for reject file (the total number of transaction lines will be different for the reject file).

format_header_fields()

parse THEAD line into variables

format_tvats_fields()

parse TVATS line into variables

format_detail_fields()

parse TDETL line into variables

Things that must be validated in validate_THEAD, validate_TVATS, and validate_TDETL: (if validation fails, reject transaction but continue processing file)

- item valid (check desc_look) (If only given UPC, get SKU from upc_ean)
- partner/partner type valid if given (use validate_partner_id, and check against partner table)
- merchandise invoices cannot be associated with a partner, and must have supplier
- credit notes from a partner cannot have item records attached to it unless partner type is in (S1,S2,S3).
- order number valid if given (check ordhead)
- reference invoice id valid if given (check invc_head)
- RTV order number valid if given (check against rtv_head)
- if total qty given, individual item qtys sum to total (only need to check this if invc_match_qty_ind is 'Y' on system_options)
- total merchandise cost on THEAD line matches sum of costs from TDETL lines (sum of unit cost *qty)
- Credit notes cannot have both a reference invoice id and a reference RTV
- If reference invoice and reference RTV are null, then no terms are required. If they are included, set due date to later of system date and sup_invoice_date (use set_due_date).
- If SKU is not given, must have UPC
- dates are valid dates, numbers are all numeric
- If terms not given, default to supplier terms (from sups table)
- terms must exist on terms table
- VAT code must exist on vat_codes (validate on THEAD, and TVATS lines—use a separate function validate_VAT for this and pass in the VAT code)
- Addr_key must exist in the addr table for the partner or supplier given, if not given, will default to primary address of partner or supplier

- Paid_ind needs to be Y or N, if not given will default to that on the sups table corresponding to the supplier. If only a partner is given, paid_ind will default to N.

validate_THEAD()

Validate THEAD line variables as needed

validate_TVATS()

Validate VAT code from TVATS line.

validate_TDETL()

validate TDETL line variables as needed.

If UPC is given but SKU isn't, get the SKU from the upc_ean table

validate_order(order_no)

Check that a given order no exists on the ordhead table

validate_shipment(ship_no)

Check that a given shipment number exists on the shipment table

validate_carton(ship_no, carton_no)

Check that the given ship_no/carton_no exists on the shipsku table

validate_vatcode(vat_code)

Check that a given VAT code exists on the vat_codes table

validate_partner_type

Check that given partner_type is legal.

validate_addr_key

Check that given addr_key is legal.

get_supplier_terms

Fetch terms from the sups table for the given supplier, if terms not given in file.

get_addr_key

Fetch addr_key from the addr table, if not given in file.

get_paid_ind

Fetch paid_ind from the sups table, if not given in file.

set_due_date()

Sets due date to later of system date and sup_inv_date.

validate_terms()

Check that terms exists on terms table.

insert_header()

Insert a record into the `invc_head` table.

A credit memo is treated as matched from the moment it is created (since it must refer to another invoice or a return to vendor number). Regular merchandise invoices and non-merchandise invoices are created in unmatched status.

Need to check for duplicate invoices (duplicate supplier invoice number/(supplier)/(partner type/partner id)/supplier invoice date) . If a duplicate is found, reject the duplicate transaction, but continue processing the rest of the file.

The due date should be calculated as follows if terms are given:

Select duedays from the terms table for the terms from the file. Due_date = invoice date + duedays.

The terms discount percent should be taken from the terms table if it is not in the file.

Values to insert into `invc` head should be as follows:

INVC_ID from invc_sql.next_invc_id

INVC_TYPE 'I' for merchandise invoice, 'C' for credit note, 'N' for non-merchandise invoice (from file)

SUPPLIER supplier from file

EXT_REF_NO supplier invoice id from file

STATUS 'U' for merchandise and non merchandise invoices, 'M' for credit invoices

EDI_INVC_IND 'Y'

MATCH FAIL IND 'N'

REF INVC ID null for invoice, reference invoice number for credit note if given

REF_RTV_ORDER_NO	null for invoice, RTV order number for credit note if given
REF_RSN_CODE	null
TERMS	terms from file (or terms table if not given in file)
DUE_DATE	calculated as above
PAYMENT_METHOD	from file
TERMS_DSCNT_PCT	calculate as above
TERMS_DSCNT_APPL_IND	from file
FREIGHT_TERMS	from file
CREATE_	'EDIUPINV'
CREATE_DATE	vdate
INVC_DATE	from file
MATCH_ID	(null)
MATCH_DATE	(null)
APPROVAL_ID	(null)
APPROVAL_DATE	(null)
FORCE_PAY_IND	'N'
FORCE_PAY_ID	(null)
POST_DATE	(null)
CURRENCY_CODE	from file
EXCHANGE_RATE	from file if given, null otherwise
TOTAL_MERCH_COST	from file
TOTAL_QTY	from file if given
EDI_SENT_IND	'N'
COMMENTS	(null)
PARTNER_TYPE	from file

PARTNER_ID	from file
ADDR_KEY otherwise	from file if given, addr table
PAID_IND table otherwise	from file if given, sups
insert_detail()	

Insert a record into the invc_detail table.

Default invoice unit cost to original unit cost , invoice qty to original qty, invoice vat rate to original vat rate (“original” comes from the invoice file)

Values should be as follows:

INVC_ID	created by program
SKU	from file
UPC	only if given in file
UPC_SUPPLEMENT	only if given in file
INVC_UNIT_COST	original cost from file
INVC_QTY	original qty from file
INVC_VAT_RATE	original vat rate from file (if given)
STATUS which will be ‘M’)	‘U’ (except for credit memo,
ORIG_UNIT_COST	from file
ORIG_QTY	from file
ORIG_VAT_RATE	from file
COST_DSCRPNY_IND	‘N’
QTY_DSCRPNY_IND	‘N’
VAT_DSCRPNY_IND	‘N’
PROCESSED_IND	‘N’
COMMENTS	(null)
insert_wksht()	

Insert a record into the `invc_match_wksht` table.

Values should be as follows:

<code>INVC_ID</code>	created by program
<code>SKU</code>	from file
<code>INVC_UNIT_COST</code>	from file
<code>SHIPMENT</code>	from file
<code>CARTON</code>	from file
<code>INVC_MATCH_QTY</code>	(null)

`insert_order_xref(order_no)`

(Only call this function if an order number was given—only call once for each distinct order number for each invoice)

Call `INVC_SQL.APPLY_XREF`, which will insert the order number and all received shipments with that order number into the `invc_xref` table. The `apply_to_future_ind` parameter should be 'Y'.

`insert_ship_xref(shipment)`

Make sure the shipment has been received. (If not, reject transaction). Check to make sure that a record with this shipment and invoice does not already exist on the `invc_xref` table; if it does not, insert (`apply_to_future_ind` should be 'N', everything else except `invoice_id` and `shipment` will be null).

`insert_match_queue()`

Insert a record into the `invc_match_queue` table with the new invoice ID. (Don't call this function for credit memos; they don't need to go into the match queue.)

`insert_non_merch()`

If a sales tax is given on the header line, insert it into the `invc_non_merch` table (with a `non_merch_code` of 'T'); if a freight charge is given, insert it into `invc_non_merch` with a `non_merch_code` of 'F'; if a miscellaneous charge is given, insert with a code of 'M'. If VAT codes are given, insert them.

`insert_merch_vat()`

Insert a record into the `invc_merch_vat` table. Processing should continue with no error if duplicate values are found.

`write_to_reject()`

Write record to reject file (use linked list)

get_line()

Read in a line from input file and determine the line type.

reset_variables()

Set counters back to zero.

final()

Restart/recovery close

Close files, remove temporary file.

I/O Specification

Input file format:

FHEAD (start of file)

THEAD (Invoice level info)

TVATS (Summary level –cost at a VAT rate)

TDETL (items within an invoice)

TSHIP (shipment and carton numbers within an invoice)

Note: All TSHIP records for a single item must follow immediately after the TDETL record.

TTAIL (marks end of invoice)

FTAIL (marks end of file)

An input file can contain multiple invoices.

Reject file will have an identical format. If no records are rejected, it will consist of only the FHEAD and FTAIL lines.

All character variables should be right-padded with blanks and left justified; all numerical variables should be left-padded with zeroes and right-justified. Missing variables should be blank.

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	File head descriptor	Char(5)	FHEAD	Describes file line type
	Line id	Char(10)	0000000001	Sequential file line number
	Gentran ID	Char(5)	UPINV	Identifies which translation Gentran uses
	Current date	Char(14)		File date in YYYYMMDDHH24MISS format
THEAD	File record descriptor	Char(5)	THEAD	Start of an invoice transaction
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Transaction type flag	Char(1)		I=Invoice C=Credit note N= non-merchandise invoice
	Supplier invoice number	Char (30)		Supplier's identifier for this invoice
	Cross-reference invoice number	Number(10)		Invoice that a credit note is for (blank for an invoice)
	Cross-reference RTV number	Number(6)		RTV order that a credit note is for (blank for an invoice)
	Partner Type	Char(6)		BK=Bank AG=Agent FF=Freight Forwarder IM=Importer BR=Broker FA=Factory AP=Applicant CO=Consolidator CN=Consignee S1=Supplier hierarchy level 1 (e.g. manufacturer) S2=Supplier hierarchy level 2 (e.g. distributor) S3=Supplier hierarchy level 3 (e.g. wholesaler)
	Partner ID	Char(10)		Credit note or non-merchandise invoice partner.

Record Name	Field Name	Field Type	Default Value	Description
	Supplier	Number(10)		Invoice supplier
	Supplier invoice date	Char(14)		Inv_head.inv_date YYYYMMDDHH24MISS Date invoice was issued by supplier
	Terms	Char(15)		Inv_head.terms If not given in file, take from supplier terms
	Terms discount percentage	Number(12)		Discount percent if invoice paid on time 4 implied decimal places (optional— if not in file, take from terms table)
	Discount apply indicator	Char(1)		‘Y’—discount HAS been applied to total_merch_cost ‘N’—it has not
	Discount apply non merch ind	Char(1)		‘Y’—discount HAS been applied ‘N’—it has not
	Payment method	Char(6)		Inv_head.payment_method LC=letter of credit OA= Open Account WT= wire transfer (other values possible)
	Currency code	Char(3)		Currency code of invoice currency
	Exchange rate	Number(20)		From the invoice currency to the primary currency. 10 implied decimal places. Optional.
	Total merchandise cost	Number(20)		Total invoice cost, including all items on invoice (4 implied decimal places) (NOT including tax) VAT EXCLUSIVE
	Total quantity	Number(12)		Total quantity of items on invoice (4 implied decimal places)
	Order number 1	Number(8)		(Retek) Order number associated with the invoice
	Order number 2	Number(8)		2 nd order associated w/invoice if exists
	Order number 3	Number(8)		3 rd “”
	Order number 4	Number(8)		4 th “”
	Order number 5	Number(8)		5 th “”

Record Name	Field Name	Field Type	Default Value	Description
	Shipment number 1	Number(10)		(Retek) shipment number associated w/invoice
	Shipment number 2	Number(10)		2 nd shipment associated w/invoice if exists
	Shipment number 3	Number(10)		3 rd “”
	Shipment number 4	Number(10)		4 th “”
	Shipment number 5	Number(10)		5 th ”
	Freight terms	Char(2)		Freight terms associated w/invoice
	Sales tax	Number(20)		Sales tax on invoice (4 implied decimal places) Optional
	Sales tax VAT code	char(6)		VAT code for sales tax (optional)
	Freight charges	Number(20)		Freight charges for invoice (4 implied decimal places) Optional
	Freight charge VAT code	char(6)		VAT code for freight charges (optional)
	Miscellaneous charge	Number(20)		4 implied decimal places Optional
	Misc. charge VAT code	char(6)		VAT code for miscellaneous charge—optional
	Address key	Number(6)		Address key for the supplier or partner. (optional)
	Paid_ind	Char(1)		Paid index for the invoice. ‘Y’ or ‘N’. (optional)
TVATS	File record descriptor	Char(5)	TVATS	Marks costs at VAT rate line
	Line id	Char(10)		Sequential file line number

Record Name	Field Name	Field Type	Default Value	Description
	Transaction number	Number(10)		Sequential transaction number
	VAT code	Char(6)		VAT code that applies to cost
	Cost at this VAT code	Number(20)		4 implied decimal places
TDETL	File record descriptor	Char(5)	TDETL	Marks item info line
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	SKU	Number(8)		Sku (optional but at least one of Sku/UPC must be present)
	UPC	Char(13)		Upc (optional)
	UPC-supplement	Number(5)		Upc_supplement (optional but required if UPC is given)
	Original Invoice quantity	Number(12)		How many of this item on invoice (4 implied decimal places)
	Original Unit cost	Number(20)		Unit cost of item as reported by invoice 4 implied decimal places
	Original VAT rate	Number(20)		VAT rate for this item (10 implied decimal places)
TSHIP	File record descriptor	Char(5)	TSHIP	Marks shipment/carton info line
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Shipment number	Number(10)		(Retek) shipment number associated w/ invoice
	Carton number	Char(20)		Carton ID if exists
TTAIL	File record descriptor	Char(5)	TTAIL	Marks end of transaction
	Line id	Char(10)		Sequential file line number
	Transaction number	Number(10)		Sequential transaction number
	Transaction lines	Number(6)		Total number of TDETL, TVATS, and TSHIP lines in transaction

Record Name	Field Name	Field Type	Default Value	Description
FTAIL	File record descriptor	Char(5)	FTAIL	Marks end of file
	Line id	Char(10)		Sequential file line number
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

Technical Issues

The VAT rate column on invc_detail may be changed to a VAT code column, at which point this program would need to be modified to read in a VAT code rather than a VAT rate.

Forecasted Unit Sales Upload [fcslupld]

Design Overview

The Forecasted Unit Sales Upload (fcslupld) is designed to load forecasted unit sales/issues data at the SKU location week level. The data will be loaded into the ITEM_FORECAST_XX table (where XX is the domain) by writing the data to a text file that will be loaded into the table using SQL loader.

Each forecasted sales/issues record will be validated before it is written to the text file. The validation performed will be as follows:

- Validate that the date of the record is greater than or equal to the current end of week (EOW) date and that it is a valid EOW date.
- No validation is made against the SKUs or locations themselves. It is assumed that the SKUs and locations (and their relationships) in the input file are valid in the RMS.

If the record passes validation, it will be written to a SQL loader text file.

Since one or more files may be present due to multiple domains, these files will be run in separate threads determined by the domain ID.

Scheduling Constraints

Processing Cycle:	Weekly
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	Domain ID

Restart Recovery

The logical unit of work is SKU and store records. The commit should occur every 10,000 records.

A driving cursor will not exist since this is a file-based processing module.

Program Flow

N/A

Shared Modules

C454_TO_CAL and CAL_TO_454 are called to verify that the calendar date passed in the input file into an end of week date.

Function Level Description

Threading set-up:

Since threads are determined by the value of the domain ID, the restart_program_status table should contain a row for each domain ID. The thread value of the domain ID should be used as the thread value on this table. The total number of domains/number of threads should be equal to the number of rows on the restart_program_status table. This value must be entered into the restart_control table num_threads field. Note that anytime a new domain is created that an additional row should be added to the restart_program_status table with the thread value equal to the domain ID and the restart_control table num_threads field must be incremented to equal the total number of domains.

The batch program should perform the following steps:

- Extract login ID, password from input arguments.
- Extract thread value from input file name. The input file will be fcsdupld.*nn* where *nn* will equal the domain (i.e. the thread value).
- Read first record into local variables.
- The procedures CAL_TO_454 and C454_TO_CAL should be used to verify that the date passed to the program is a valid EOW date. If not, reject the record and write error to sqllog, date must be a valid end of week date.
- Validate that the end of week date is greater than or equal to the current EOW date. If not, reject the record and write error to sqllog, date must be greater than or equal to the current eow date.
- Write each record to the sql loader file (fcsales.dat.*nn*). Files will be comma delimited.
- After all the records have been processed, write the domain to the forecast_rebuild table (if it already exists, continue). The Forecast Rebuild program will run against this table to determine which domains have to be rebuilt.

I/O Specification

Input file fcslupld.*nn* (where *nn* is the domain ID)

SKU	Numeric	8	
Location	Numeric	4	
Date	Date	8	(yyyymmdd)
Sales	Numeric	12	(12,4 – decimal implied)
Std. Dev.	Numeric	12	(12,4 – decimal implied)

Example:

10000039100320000227000000245997000000120000

Output file fcsales.dat.*nn* (comma delimited, where *nn* is the domain ID)

SKU	Numeric	8	
Location	Numeric	4	
Eow date	Date	8	(yyyymmdd)
Forecast Sales	Numeric	12,4	99999999.9999
Std. Dev.	Numeric	12,4	99999999.9999
Domain	Numeric	2	

Example:

10000039,1003,20000227,24.5997,12.0000,1

Technical Issues

N/A

Forecasting/Replenishment [Fcstprg]

Design Overview

The forecast sales information at the item and heirarchy level is no longer stored on history tables. In order to speed the performance of the data loads into RMS and subsequent hierarchy forecast builds, the tables have been built either as partitioned views or as Oracle 8 partitioned tables. The data objects are all partitioned by domain.

When new forecasts are created and loaded into RMS from RDF, the entire domain will be re-built and re-loaded. For this reason, a purge program will be used to truncate the partitions that are being re-loaded. This module will take the domain value that is being rebuilt and will truncate the partitions of the tables using dynamic sql.

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: before fcslupld.

Scheduling Diagram:

Pre-Processing:

Post-Processing:

Threading Scheme: threaded by passed parameter.

Restart Recovery

N/A

Function Level Description

Main function

This function will require two input parameters additional to the normal parameter. The domain value, and the table owner will be passed into the function along with the username/password string.

Process()

The truncate_table function should be called 4 times, each time passing the table name to be truncated, along with the table owner name. A character string should be locally declared to hold the table names. The four tables will be named:

- item_forecast_X
- dept_sales_forecast_X
- class_sales_forecast_X
- subclass_sales_forecast_X

where X is the domain value parameter.

Truncate_table function.

This function will create a truncate table command as a character string and will use dynamic sql to truncate the table. The truncate table command should indicate which table is to be truncated and the table owner value. Both of these values will be passed as parameters to the program.

I/O Specification

N/A

Forecasted Unit Sales Rollup [fcstrbld]

Design Overview

Fcstrbld.pc is designed to roll-up new or updated forecasted unit sales data from the staple SKU store history table and the fashion SKU store history table. This data will be summarized into the subclass, class and department level sales history tables.

Scheduling Constraints

Processing Cycle:	Weekly
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	Post processing to truncate forecast_rebuild table
Threading Scheme:	Department

Restart Recovery

The logical unit of work is dept, class. The commit (update subclass_sales_hist table) should occur every 10,000 records.

Driving cursor should be as follows:

```
SELECT fr.dept,
       fr.class,
       fr.subclass,
       fr.store,
       to_char(fr.eow_date, 'YYYYMMDDHH24MISS'),
       to_char(fr.eow_date, 'DD'),
       to_char(fr.eow_date, 'MM'),
       to_char(fr.eow_date, 'YYYY'),
       ';' || to_char(fr.dept) ||
       ';' || to_char(NVL(fr.class,0)) ||
       ';' || to_char(NVL(fr.subclass,0))
FROM forecast_rebuild fr,
     v_restart_dept vrd
WHERE vrd.driver_value = fr.dept
      AND vrd.driver_name = :ov_restart_driver_name
      AND vrd.num_threads = :oi_restart_num_threads
      AND vrd.thread_val = :oi_restart_thread_val
      AND (fr.dept > NVL(:ov_restart_dept, -999)
           OR (fr.dept = :ov_restart_dept
              AND (NVL(fr.class,0) > :ov_restart_class
                   OR (NVL(fr.class,0) = :ov_restart_class
                      AND (NVL(fr.subclass,0) > :ov_restart_subclass)
                         )
                 )
            )
      )
      )
      )
```

```
)  
  
ORDER BY fr.dept,  
  
fr.class,  
  
fr.store,  
  
fr.eow_date;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

init()

Initialize restart/recovery

Initialize insert and update arrays for subclass_sales_hist and class_sales_hist.

process()

Fetch driving cursor

Loop for each record returned.

Store values from cursor in local variables.

Fetch the sum (across all SKUs) of forecast_sales for the subclass.

Increment variables to hold running forecast_sales total for dept and class.

Fetch driving cursor

Call process_subclass().

If dept/class/store/eow_date changes, call process_class().

If dept/store/eow_date changes, call process_dept().

Call insert_class_struct().

Call update_class_struct().

Call insert_subclass_struct().

Call update_subclass_struct().

process_subclass()

Fetch previous forecast_sales value from subclass_sales_hist

If fetch is not found (no record exists), add values to subclass_insert array

If fetch is found,

Add values to subclass_update array

Increment delta variable to hold running total for difference in subclass' forecast_sales and updated forecast_sales.

process_class()

Fetch rowid from class_sales_hist

If fetch is not found, add values to class_insert array

If fetch is found,

Add values to class_update array

Increment delta variable to hold running total for difference in class' forecast_sales by adding delta for subclass' forecast_sales.

Reset subclass delta variable and class running total variable.

process_dept()

Perform update of dept_sales_hist

If update is not found (record does not exist), perform insert into dept_sales_hist

Reset class delta variable and dept running total variable.

insert_class_struct()

Array insert for class_sales_hist.

update_class_struct()

Array update for class_sales_hist.

insert_subclass_struct()

Array insert for subclass_sales_hist

update_subclass_stuct()

Array update for subclass_sales_hist

I/O Specification

N/A

Technical Issues

N/A

[fdaydnld.pc]

Design Overview

This module will query the store sales and warehouse issues histories for forecastable items as frequently as once a day. Four flat files for use in RDF modules will be created based on this data: one each for regular, promotion and clearance sales, and a single file for warehouse issues.

Stored Procedures / Shared Modules (Maintainability)

NA

Input Specifications

Table-To-Table

This program contains six distinct driving cursors, three for store sales (one each at department, class, and subclass domain levels), and three for warehouse issues (one each at department, class, and subclass domain levels). Only one cursor from each group of three (based on the system_options.domain_level value) will actually be used in a given instance of the program.

Store Sales Driving Cursors:

if the domain_type = 'D', then the driving cursor is as follows:

```
SELECT tdh.store,
       tdh.sku,
       tdh.tran_date,
       SUM(tdh.units),
       tdh.sales_type
FROM tran_data_history tdh,
     win_skus wsk,
     win_store wst,
     domain_dept dd
WHERE wsk.forecast_ind = 'Y'
      AND tdh.tran_code = 1 /* sales tran_code */
```

```

AND tdh.store      = wst.store

AND tdh.sku        = wsk.sku

AND tdh.sku        = wst.sku

AND ((tdh.tran_date > NVL(wst.last_sales_export_date, tdh.tran_date - 1)
      AND dd.load_sales_ind = 'N')
      OR (dd.load_sales_ind = 'Y'))

AND wsk.dept       = dd.dept

AND dd.domain_id   = :g_i_restart_thread_val

AND (tdh.sku > NVL(:ps_restart_sku,-999) OR
      (tdh.sku = :ps_restart_sku AND
      tdh.store > :ps_restart_location))

GROUP BY tdh.store,
         tdh.sku,
         tdh.tran_date,
         tdh.sales_type

UNION ALL

SELECT tdh.store,
       tdh.sku,
       tdh.tran_date,
       SUM(tdh.units),
       tdh.sales_type

FROM tran_data_hist tdh,
     rag_skus_st rss,
     rag_style rs,
     domain_dept dd

WHERE rs.forecast_ind = 'Y'

```

```
AND tdh.tran_code = 1

AND tdh.store = rss.store

AND rss.style = rs.style

AND tdh.sku = rss.sku

AND ((tdh.tran_date > NVL(rss.last_sales_export_date, tdh.tran_date - 1)

      AND dd.load_sales_ind = 'N')

      OR (dd.load_sales_ind = 'Y'))

AND rs.dept = dd.dept

AND dd.domain_id = :g_i_restart_thread_val

AND (tdh.sku > NVL(:ps_restart_sku,-999) OR

      (tdh.sku = :ps_restart_sku AND

        tdh.store > :ps_restart_location))

GROUP BY tdh.store,

         tdh.sku,

         tdh.tran_date,

         tdh.sales_type

ORDER BY 2, 1; /* sku, store */
```

if the domain_type = 'C', then the driving cursor is as follows:

```
SELECT tdh.store,

       tdh.sku,

       tdh.tran_date,

       SUM(tdh.units),

       tdh.sales_type

FROM tran_data_history tdh,

     win_skus wsk,

     win_store wst,
```



```
        domain_class dc

WHERE wsk.forecast_ind = 'Y'

AND tdh.tran_code = 1 /* sales tran_code */

AND tdh.store = wst.store

AND tdh.sku = wsk.sku

AND tdh.sku = wst.sku

AND ((tdh.tran_date > NVL(wst.last_sales_export_date, tdh.tran_date - 1)

        AND dc.load_sales_ind = 'N')

        OR (dc.load_sales_ind = 'Y'))

AND wsk.dept = dc.dept

AND wsk.class = dd.class

AND dc.domain_id = :g_i_restart_thread_val

AND (tdh.sku > NVL(:ps_restart_sku,-999) OR

        (tdh.sku = :ps_restart_sku AND

        tdh.store > :ps_restart_location))

GROUP BY tdh.store,

        tdh.sku,

        tdh.tran_date,

        tdh.sales_type

UNION ALL

SELECT tdh.store,

        tdh.sku,

        tdh.tran_date,

        SUM(tdh.units),
```

```
        tdh.sales_type
FROM tran_data_hist tdh,
        rag_skus_st rss,
        rag_style rs,
        domain_class dc
WHERE rs.forecast_ind = 'Y'
      AND tdh.tran_code = 1
      AND tdh.store = rss.store
      AND rss.style = rs.style
      AND tdh.sku = rss.sku
      AND ((tdh.tran_date > NVL(rss.last_sales_export_date, tdh.tran_date - 1)
            AND dc.load_sales_ind = 'N')
            OR (dc.load_sales_ind = 'Y'))
      AND rs.dept = dc.dept
      AND rs.class = dc.class
      AND dc.domain_id = :g_i_restart_thread_val
      AND (tdh.sku > NVL(:ps_restart_sku,-999) OR
            (tdh.sku = :ps_restart_sku AND
             tdh.store > :ps_restart_location))
GROUP BY tdh.store,
        tdh.sku,
        tdh.tran_date,
        tdh.sales_type
ORDER BY 2, 1; /* sku, store */
```

If the domain_type = 'S', the driving cursor is as follows:

```
SELECT tdh.store,
```

```

        tdh.sku,
        tdh.tran_date,
        SUM(tdh.units),
        tdh.sales_type
FROM tran_data_history tdh,
        win_skus wsk,
        win_store wst,
        domain_subclass ds
WHERE wsk.forecast_ind = 'Y'

AND tdh.tran_code = 1 /* sales tran_code */

AND tdh.store = wst.store

AND tdh.sku = wsk.sku

AND tdh.sku = wst.sku

AND ((tdh.tran_date > NVL(wst.last_sales_export_date, tdh.tran_date - 1)

        AND ds.load_sales_ind = 'N')

        OR (ds.load_sales_ind = 'Y'))

AND wsk.dept = ds.dept

AND wsk.class = ds.class

AND wsk.subclass = ds.subclass

AND ds.domain_id = :g_i_restart_thread_val

AND (tdh.sku > NVL(:ps_restart_sku,-999) OR

        (tdh.sku = :ps_restart_sku AND

                tdh.store > :ps_restart_location))

GROUP BY tdh.store,

        tdh.sku,

        tdh.tran_date,

```

tdh.sales_type

UNION ALL

```
SELECT tdh.store,
       tdh.sku,
       tdh.tran_date,
       SUM(tdh.units),
       tdh.sales_type
FROM   tran_data_hist tdh,
       rag_skus_st rss,
       rag_style rs,
       domain_subclass ds
WHERE  rs.forecast_ind = 'Y'
       AND tdh.tran_code = 1
       AND tdh.store = rss.store
       AND rss.style = rs.style
       AND tdh.sku = rss.sku
       AND ((tdh.tran_date > NVL(rss.last_sales_export_date, tdh.tran_date - 1)
            AND ds.load_sales_ind = 'N')
            OR (ds.load_sales_ind = 'Y'))
       AND rs.dept = ds.dept
       AND rs.class = ds.class
       AND rs.subclass = ds.subclass
       AND ds.domain_id = :g_i_restart_thread_val
       AND (tdh.sku > NVL(:ps_restart_sku,-999) OR
```

```

        (tdh.sku = :ps_restart_sku AND
         tdh.store > :ps_restart_location))
GROUP BY tdh.store,
         tdh.sku,
         tdh.tran_date,
         tdh.sales_type
ORDER BY 2, 1; /* sku, store */

```

Warehouse Issues Driving Cursor:

if the domain_type = 'D', the driving cursor is as follows:

```

SELECT tdh.wh,
       tdh.sku,
       tdh.tran_date,
       SUM(tdh.units)
FROM tran_data_history tdh,
     win_skus wsk,
     win_wh ww,
     domain_dept dd
WHERE wsk.forecast_ind = 'Y'
      AND tdh.tran_code = 32 /* issues tran_code */
      AND tdh.wh = ww.wh
      AND tdh.sku = wsk.sku
      AND tdh.sku = ww.wh.sku
      AND ((tdh.tran_date > NVL(ww.last_issues_export_date, tdh.tran_date - 1)
           AND dd.load_sales_ind = 'N')
           OR (dd.load_sales_ind = 'Y'))
      AND wsk.dept = dd.dept

```

```
AND dd.domain_id    = :g_i_restart_thread_val
AND (tdh.sku > NVL(:ps_restart_sku,-999) OR
     (tdh.sku = :ps_restart_sku AND
      tdh.wh > :ps_restart_location))
GROUP BY tdh.wh,
         tdh.sku,
         tdh.tran_date

UNION ALL

SELECT tdh.wh,
       tdh.sku,
       tdh.tran_date,
       SUM(tdh.units)
FROM   tran_data_hist tdh,
       rag_skus_wh rsw,
       rag_style rs,
       domain_dept dd
WHERE  rs.forecast_ind = 'Y'
AND    tdh.tran_code    = 32 /* issues tran_code */
AND    tdh.wh           = rsw.wh
AND    rsw.style        = rs.style
AND    tdh.sku          = rsw.sku
AND    ((tdh.tran_date > NVL(rsw.last_issues_export_date, tdh.tran_date - 1)
        AND dd.load_sales_ind = 'N')
OR     (dd.load_sales_ind = 'Y'))
```

```

AND rs.dept      = dd.dept

AND dd.domain_id = :g_i_restart_thread_val

AND (tdh.sku > NVL(:ps_restart_sku,-999) OR
     (tdh.sku = :ps_restart_sku AND
      tdh.store > :ps_restart_location))

GROUP BY tdh.wh,

        tdh.sku,

        tdh.tran_date

ORDER BY 2, 1; /* sku, wh */

If the domain_type = 'C', the driving cursor is as follows:

SELECT tdh.wh,

        tdh.sku,

        tdh.tran_date,

        SUM(tdh.units)

FROM tran_data_history tdh,

        win_skus wsk,

        win_wh ww,

        domain_class dc

WHERE wsk.forecast_ind = 'Y'

AND tdh.tran_code = 32 /* issues tran_code */

AND tdh.wh      = ww.wh

AND tdh.sku     = wsk.sku

AND tdh.sku     = ww.wh.sku

AND ((tdh.tran_date > NVL(ww.last_issues_export_date, tdh.tran_date - 1)

      AND dc.load_sales_ind = 'N')

      OR (dc.load_sales_ind = 'Y'))

```

```
AND wsk.dept      = dc.dept

AND wsk.class     = dc.class

AND dc.domain_id  = :g_i_restart_thread_val

AND (tdh.sku > NVL(:ps_restart_sku,-999) OR

     (tdh.sku = :ps_restart_sku AND

      tdh.wh > :ps_restart_location))

GROUP BY tdh.wh,

         tdh.sku,

         tdh.tran_date

UNION ALL

SELECT tdh.wh,

       tdh.sku,

       tdh.tran_date,

       SUM(tdh.units)

FROM   tran_data_hist tdh,

       rag_skus_wh rsw,

       rag_style rs,

       domain_class dc

WHERE  rs.forecast_ind = 'Y'

AND    tdh.tran_code = 32 /* issues tran_code */

AND    tdh.wh        = rsw.wh

AND    rsw.style     = rs.style

AND    tdh.sku       = rsw.sku

AND    ((tdh.tran_date > NVL(rsw.last_issues_export_date, tdh.tran_date - 1)
```



```

        AND dc.load_sales_ind = 'N')
    OR (dc.load_sales_ind = 'Y'))
    AND rs.dept      = dc.dept
    AND rs.class     = dc.class
    AND dc.domain_id = :g_i_restart_thread_val
    AND (tdh.sku > NVL(:ps_restart_sku,-999) OR
        (tdh.sku = :ps_restart_sku AND
         tdh.store > :ps_restart_location))
GROUP BY tdh.wh,
        tdh.sku,
        tdh.tran_date
ORDER BY 2, 1; /* sku, wh */

```

if the domain_type = 'S', the driving cursor is as follows:

```

SELECT tdh.wh,
        tdh.sku,
        tdh.tran_date,
        SUM(tdh.units)
FROM tran_data_history tdh,
        win_skus wsk,
        win_wh ww,
        domain_subclass ds
WHERE wsk.forecast_ind = 'Y'
    AND tdh.tran_code = 32 /* issues tran_code */
    AND tdh.wh      = ww.wh
    AND tdh.sku     = wsk.sku
    AND tdh.sku     = ww.wh.sku

```

```
AND ((tdh.tran_date > NVL(wwh.last_issues_export_date, tdh.tran_date - 1)
      AND ds.load_sales_ind = 'N')
      OR (ds.load_sales_ind = 'Y'))
AND wsk.dept      = ds.dept
AND wsk.class     = ds.class
AND wsk.subclass  = ds.subclass
AND ds.domain_id  = :g_i_restart_thread_val
AND (tdh.sku > NVL(:ps_restart_sku,-999) OR
      (tdh.sku = :ps_restart_sku AND
        tdh.wh > :ps_restart_location))
GROUP BY tdh.wh,
         tdh.sku,
         tdh.tran_date

UNION ALL

SELECT tdh.wh,
       tdh.sku,
       tdh.tran_date,
       SUM(tdh.units)
FROM   tran_data_hist tdh,
       rag_skus_wh rsw,
       rag_style rs,
       domain_subclass ds
WHERE  rs.forecast_ind = 'Y'
       AND tdh.tran_code = 32 /* issues tran_code */
```

```

AND tdh.wh      = rsw.wh

AND rsw.style    = rs.style

AND tdh.sku      = rsw.sku

AND ((tdh.tran_date > NVL(rsw.last_issues_export_date, tdh.tran_date - 1)
      AND ds.load_sales_ind = 'N')
      OR (ds.load_sales_ind = 'Y'))

AND rs.dept      = ds.dept

AND rs.class     = ds.class

AND rs.subclass  = ds.subclass

AND ds.domain_id = :g_i_restart_thread_val

AND (tdh.sku > NVL(:ps_restart_sku,-999) OR
      (tdh.sku = :ps_restart_sku AND
      tdh.store > :ps_restart_location))

GROUP BY tdh.wh,

        tdh.sku,

        tdh.tran_date

ORDER BY 2, 1; /* sku, wh */

```

Output Specifications

The four output files will be named as follows:

- Regular sales: rdsalnn.dat
- Promotion sales: pdsalnn.dat
- Clearance sales: cdsalnn.dat
- Warehouse issues: dlyissnn.dat

where nn is a two-digit number corresponding to the domain_id from which the data was derived, e.g. the regular sales output file from data in domain 13 would be named rdsal13.dat, the warehouse issue output file from data in domain 8 would be named dlyiss08.dat, and so on.

Output File Format:

Field Name	Field Type	Description
Location	Char(20)	Store or Warehouse ID. Left-justified.
SKU	Char(20)	SKU. Left-justified.
Date	Char(8)	Transaction date. Left-justified. (‘YYYYMMDD’)
Quantity	Char(13)	Sales/Issues. Contains up to four decimal places. If value is fractional, the quantity will be printed with decimal character ‘.’ included, e.g. “1023.25”. Right-justified.

Function Level Description

main():

The standard Retek main() function. Calls init(), process(), and final().

init():

Initialize restart recovery by calling retek_init() and set up the output files. The files should be named as described in Output Specifications. init() will also call a cursor to get the vdate from the period table and the domain_level from system_options.

format_buffer():

Formats the string which will be used to write to the output file.

init_arrays():

Allocates the necessary memory (sized to the restart_max_counter) to the arrays used in processing.

process():

This function makes a call to format_buffer() and init_arrays(). A restart flag indicating whether or not the sales loop was completed should be checked; if the loop did not complete (or this is the program’s initial run), process_sales() is called, after which changes are committed and process_issues() is called, once complete, changes are committed. If the restart flag indicates that sales processing was complete at restart, skip process_sales() and call process_issues(), committing after completion.

process_sales():

This function contains the three sales driving cursors. Use the `domain_level` value fetched in `init()` to determine which cursor to use.

Open the appropriate driving cursor. In a while loop, perform an array fetch. If the cursor is exhausted, set a flag to indicate this. Use a for loop to loop through each of the records in the array. For each, call `write_daily_sales()`, passing the record's `sales_type`, `store`, `SKU`, `tran_date` and `units` as parameters. After each set of date is processed by the for loop, make a call to `retek_force_commit()`. If the exhausted cursor flag is set, break out of the while loop.

process_issues():

This function contains the three issues driving cursors. Use the `domain_level` value fetched in `init()` to determine which cursor to use. Open the appropriate driving cursor. In a while loop, perform an array fetch. If the cursor is exhausted, set a flag to indicate this. Use a for loop to loop through each of the records in the array. For each, call `write_daily_issues()`. After each set of date is processed by the for loop, make a call to `retek_force_commit()`. If the exhausted cursor flag is set, break out of the while loop.

write_daily_sales():

This function takes five strings as parameters: `sales_type`, `store`, `SKU`, `tran_date`, and `units`. If the `sales_type` is equal to "R" (regular sale), then write the data to the `rdsalnn.dat` file. If equal to "P" (promotion sale), write to the `pdsalnn.dat` file. If equal to "C" (clearance sale), write to the `cdsalnn.dat` file. All output will be written to files using the string formatted in `format_buffer()`.

write_daily_issues():

This function takes four strings as parameters: `wh`, `SKU`, `tran_date`, and `units`. Write this data to the `dlyissnn.dat` file using the string formatted in `format_buffer()`.

final():

Take care of file clean up, freeing of memory, and complete the restart recovery process by calling `retek_close()`.

Scheduling Considerations

This program should run weekly in phase 4.

It must be run before the weekly sales/issues download programs: `fisdnllds.pc`, `fisdnlidf.pc`, `fsadnlidf.pc`, or `fsadnllds.pc`.

It may also run daily in phase 4, if daily sales/issues download are necessary.

Restart/Recovery

This program will use restart recovery. The logical unit of work is each unique SKU/Location combination. The program will also maintain a restart flag which indicates whether or not the first process loop (sales) was complete at the time of restart. Rather than being included in a driving cursor's WHERE clause (as with most restart variables), this flag will only be used to determine which process loop to begin in on restart.

[fdayupld – perl]

Functional Area

Enterprise interfaces – interface daily forecasts between RMS and RDF

Module Affected

Fdayupld - Perl script to load daily forecast data into daily_item_forecast_nn.

Design Overview

The forecast sales information tables have been built either as partitioned views or as Oracle 8 partitioned tables. The data objects are all partitioned by domain. When new forecasts are created and loaded into RMS from RDF, the entire domain will be re-built and re-loaded.

This perl script will accept a daily item forecast input file from RDF and load it into the correct RMS daily_item_forecast partition using sqlloader. The correct partition will be determined by using the input file's extension. The script will create a control file to truncate and load the data from the input file into the domain's partition.

Output Specifications

'File-To-Table'

Table/View	Select	Insert (sqlload)	Truncate Table	Truncate Partition
DAILY_ITEM_FORECAST		X		X

Function Level Description

The input file will be in this format:

Input file filename.nn (where nn is the domain ID)

SKU	numeric	8	
Location	numeric	4	
Date	date	8	(yyyymmdd)
Sales decimal places)	numeric	12	(12,4 - 4 implied

Std. Dev. numeric 12 (12,4 - 4 implied
decimal places)

Example record:

10000266100220000116000000009691000000108542

sku:10000266

loc:1002

date:20000116

sales:000000009691 (or .9691 after decimals are put back in)

std dev:000000108542 (or 10.8542 after decimals are put back in)

The output control file should have the below form where NN is the domain of the input file.

LOAD DATA

TRUNCATE INTO TABLE daily_item_forecast_NN

(

ITEM position(1:8) INTEGER EXTERNAL,

LOC position(9:12) INTEGER EXTERNAL,

DATA_DATE position(13:20) DATE "YYYYMMDD",

FORECAST_SALES position(21:32) DECIMAL EXTERNAL
":FORECAST_SALES/10000",

FORECAST_STD_DEV position(33:44) DECIMAL EXTERNAL
":FORECAST_STD_DEV/10000",

DOMAIN_ID CONSTANT NN

)

The input file should be used to get the domain_id. Once the domain is retrieved, the control file should be created. Finally, the data file should be loaded into the database using the generated control file and the userid/password@database from the command line.

The script will accept two parameters, a connect string and the file name of the daily forecast input file:

fdayupld userid/passwd@database datafile

Assumption:

The files sent from RDF will contain an entire domain. Domains cannot be split between multiple files or data will be lost.

Scheduling Considerations

Fdayupld should be run in phase 2. It has no pre or post dependencies except that it should not be run during the replenishment run (phase 3).

[fifcoadn.pc]

Design Overview

This program calls a function which brings data from Retek's department, class, subclass, location staging tables into PeopleSoft's staging tables. It is missing important error handling.

Function Level Description

get_dept()

get_class()

get_subclass()

get_location()

- Currently, code does not trap for "if (SQL_ERROR_FOUND)" after the package calls. It should handle this.

Currency Interface [fifcuru2107]

Design Overview

Once the Platform Dependent Load Process (PDLP) uploads Retek's staging table `fif_currencies` and `fif_currency_rates`, a new program will read all rows on the staging tables and validate this data before inserting to or updating the `currencies` table with the new currency code information.

The `fif_currency_rates` table will contain the exchange type found in the financial package (Oracle or PeopleSoft). The program will resolve the financial package's type by reference to the new `fif_currency_xref` table, which will translate the financial type into the RMS type. The RMS type will be written to the `currency_rates` table. The `fif_currency_xref` table is populated at installation and is maintained by system administration. The table is not updateable by other users.

The `currency_rates` table will be updated with the new rates information provided that the currency is already on the `currencies` table. This is enforced by the integrity constraint `CRT_CUR_FK`. After the new information has been processed, it will immediately be deleted from the respective staging tables, `fif_currencies` and `fif_currency_rates`. If an error occurs or invalid data is found, the `currencies` and/or `currency_rates` table will not be updated, a record will be written to an error table (`if_errors`), and the row will not be deleted from the staging table(s).

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Nightly, prior to `datesys.pc`, which must run after this process because of the pending rate/vdate dependency of these processes. `Currency.pc` can be removed from the schedule as its function is performed in `fifcuru2.pc`.

Scheduling Diagram: It should be scheduled prior to `datesys.pc` in the nightly schedule.

Restart Recovery

The logical unit of work for the validation upload is each row on the `fif_currencies` table:

```
select  currency_code,  
  
        currency_desc,  
  
        currency_cost_fmt,  
  
        currency_rtl_fmt,  
  
        currency_cost_dec,  
  
        currency_rtl_dec,  
  
        rowid
```

```
      from fif_currencies;
```

and the logical unit of work for the validation upload is each row on the `fif_currency_rates` table:

```
select  distinct c.currency_code,  
  
          c.fif_exchange_type,  
  
          x.rms_exchange_type,  
  
          c.exchange_rate,  
  
          c.rowid  
  
      from fif_currency_rates c, fif_currency_xref x  
  
     where c.fif_exchange_type = x.fif_exchange_type  
  
          or c.fif_exchange_type = x.rms_exchange_type;
```

Program Flow

Platform Dependent Upload Program (PDLP)

The platform dependent loading process will select fields from the financial package's currency rate table and insert them into the staging tables, `fif_currencies` and/or `fif_currency_rates`. No validation of data will occur except field type and length. Specific to the financial package being used, certain columns must be selected to match up to the columns on the staging tables, `fif_currencies` and/or `fif_currency_rates`. This module is not part of this design, as it is platform specific.

The following assumptions will be made about the PDLP:

1. The `fif_currencies` table will only be loaded with currency information pertaining to tomorrow (`VDATE+1`).
2. Any rate information can be provided if the rate type is defined on the `fif_currency_x_ref` table.

Validation Upload (`fifcuru2.pc`)

The program will read a row from `fif_currencies` and check for its existence on currencies.

1. If it does exist, no NULL validation is performed. The currencies record is updated with the new information. If all columns are not updated, the old value for that column will be retained. Should the rate information be updated, appropriate row(s) will be written to `currency_rates`.
2. If the `fif_currencies` record does not exist on the currencies table, the program will validate each field. If a record passes all validation checks, a record will be written to currencies. As many rows as are on the staging table, `fif_currency_rates` will be written to `currency_rates` provided the rate type is defined on the `fif_currency_xref` table.

In either of the above cases, a `fif_currencies` row that has been successfully processed will be deleted and a `fif_currency_rates` row that has been successfully processed will be deleted. A row that has not been successfully processed will remain on the staging table and have appropriate error message(s) written to an error table.

Validations for `fif_currencies`:

- `currency_code` – NOT NULL
- `currency_desc` – NOT NULL
- `currency_cost_dec` – NOT NULL and must be between 0 and 4.
- `currency_rtl_dec` – NOT NULL and must be between 0 and 4.

Validations for `fif_currency_rates`:

- `currency_code` – NOT NULL
- `fif_exchange_type` NOT NULL and exists on `fif_currency_xref`
- Must exist on `fif_currencies` prior to insert.

Defaults:

Currency cost/retail format: If this format is not bridged in from the Financial Packages, it will be defaulted to a set format, with the correct precision based on the `currency_cost_dec`, `currency_rtl_dec`.

`INT_PROCESS()`

- Read record, determine if the record is an insert or an update to currencies.
- Update: perform validation, update currencies, insert `currency_rates` if necessary.
- Insert: perform validation, insert to currencies, insert to `currency_rates`.

Validation details:

- Check for uniqueness of `currency_code` and rowid combination.

Set `error_msg` := “%s1 with a currency code of %s2 has more than one record.”

If error occurs, call package `WRITE_INTERFACE_ERRORS` to write to `if_errors`. Set `valid_flag` to F.

- Check that `currency_rtl_dec` and `currency_cost_dec` are between 0 and 4. Set format mask to match format sent in if not bridged.

Set `error_msg` := “%s1 with value %s2 is not between 0 and 4”

Validate that `currency_rtl_dec` and `currency_cost_dec` are between 0 and 4.

If error occurs, call package `interface_sql.insert_interface_errors` to write to `if_errors`. Set `valid_flag` to F.

- Check that `currency_code`, `currency_desc`, `currency_cost_dec`, or `currency_rtl_dec`, are all not null. (For insert to currencies only).

Set `error_msg` := “%s1 with value %s2 is a not null field.”

If error occurs, call package `interface_sql.insert_interface_errors` to write to `if_errors`. Set `valid_flag` to F.

Update details:

- Update currencies. When inserting to currency_rates, the effective_date will always be defaulted to (VDATE+1) . If there is an error: set error_msg := “Error inserting to currency_rates for currency %1” and call package interface_sql.insert_interface_errors to write to if_errors.

Shared Modules

New error package ifsqlb.pls

Function Level Description

Individual rows not updated to the base table are retained on staging tables, and a row is written to the if_errors table.

I/O Specification

None.

Technical Issues

None.

[fifglndn1107.pc]

Design Overview

This process will extract detailed stock ledger information for certain transaction types on a daily basis in order to bridge the information to an interfaced financial application. The detailed information will retain certain reference information, such as stock adjustment reason codes, which are lost during the normal daily roll-up process (saldly.pc).

This process reads from the if_tran_data table, instead of the actual tran_data table itself. Hence, it requires the 'PRE-SALDLY' and 'PRE-FIFGLDN1' run of prepost.pc to have occurred prior to its initiation. It can run in parallel to fifglnd2.pc.

New Codes Table Information

Concerning GL data, there are two factors that need to be addressed:

1. Some transaction types/amount types need to be bridged at a low level to contain detailed reference information, while others do not.
2. There are only certain transaction types/amount types that need to be bridged for a given client. For instance, net sales at retail may be needed for one client, while net sales cost by be needed by another.

This process should allow the flexibility to bridge the required information without coding efforts. Since users can manipulate the codes detail table through a dialogue, this would be a good tool to allow this maintenance.

The new code type that will be created for this process is the General Ledger – Single Transaction 'GLST' type. This will include transaction types/amount types to go through the sku-level detailed processing if the feed to GL is to be daily. The code for this will be a 3-character field, the first two characters determining the transaction code, the third character determining if the cost or retail component of the amount is to be bridged ('C' for Cost, 'R' for Retail, and 'B' for Both). For instance, if the code 01R is on the table, the net sales retail will be bridged at the sku-detail level.

There will be two other codes types in the total GL interface: General Ledger – Rolled Transaction ‘GLRT’ and General Ledger – Monthly Transaction ‘GLMT’. These are part of the FIFGLDN2 and FIFGLDN3 designs.

Note: There are several risks in the maintenance of these codes types:

The codes table is not masked, and one could incorrectly enter an R12 instead of 12R, etc. This will result in not sending the code 12 GL transactions.

‘12R’ could be entered in more than one codes type. This will result in duplicate processing, although at different levels of roll-up. There is no cross-checking to verify that a code entered in one code type is not in the other.

Daily-level GL Feed

New processes:

The new program `fifgldn1.pc` will read the `if_tran_data` table for each transaction type/amount type in the codes table, and post it to a Retek general ledger staging table (`fif_gl_data`) at the sku detail level, complete with reference fields. These reference fields will be bridged to the financial application for segregation of stock adjustment codes, mapping transfers to transfer numbers, etc., depending on the client need.

The new program `fifgldn1.pc` can run in parallel to the `saldly` process, but must run after the `if_tran_data` load process, as it will process rows on `if_tran_data`.

The local currency will be retrieved for all locations from their corresponding store/warehouse rows.

Scheduling Constraints

Processing Cycle: Daily, possibly in parallel to `fifgldn2.pc`, after `PREPOST(PRE-SALDLY)` and `PREPOST(PRE-FIFGLDN1)`.

Pre-Processing: Prepost for `PRE-SALDLY`. Prepost for `PRE-FIFGLDN1` (this creates indexes)

Post-Processing

Threading Scheme: Based on store/wh.

Restart Recovery

Driving Cursor

This program will be fully restartable. The LUW will be based upon dept, class, subclass.

```
Select distinct
Td.sku,
    Td.Dept,
    Td.Class,
    Td.Subclass,
    Td.Store,
    Td.Wh,
    Td.Tran_date,
    Td.Tran_code,
    Td.Adj_code,
    Td.Total_cost,
    Td.Total_retail,
    Td.Ref_no_1,
    Td.Ref_no_2,
    St.currency_code,
    Wh.currency_code,
    cd.code
from   if_tran_data td,
       code_detail cd,
       store st,
       wh
       v_restart_store_wh
where (cd.code_type = 'GLST')
      and (substr(cd.code, 1, 2) = td.tran_code)
      and (td.store = st.store (+)
           and td.wh = wh.wh (+) )
```

```

and v.driver_name = :os_restart_driver_name

and v.driver_value = :td.store

and v.num_threads = :oi_restart_num_threads

and v.thread_val = :oi_restart_thread_val

and (td.dept > NVL(:ora_restart_dept, td.dept-1) OR

      (td.dept = :ora_restart_dept AND

        (td.class > :ora_restart_class OR

          (td.class = :ora_restart_class AND

            (td_subclass > :ora_restart_subclass))))))

order by      dept,

class,

subclass;

```

Program Flow

The high-level processing for this program is as follows:

1. Read rows
2. Obtain its cost component from code_detail (substr(code, 3,1)). If it is 'C' for Cost, insert a row to fif_gl_data for the cost amount. If it is 'R', insert a row for the retail amount. If it is 'B' for both, insert one row for each.
3. If the processing/insert was successful, continue, else write a row to the if_errors interface error table.

init() -

- Initialize restart recovery.
- Set up array structures for the driving cursor fetch as well as one for inserts into the FIF_GL_DATA table (dept, class, subclass, location, loc_type, tran_date, tran_code, adj_code, amount, cost_retail_flag, ref_no_1, ref_no_2, sku, currency_code in the structure).
- Normal array sizing should occur [size_arrays()]

process() -

- open the driving cursor

- WHILE(1)
 - fetch g_l_restart_max_counter records from driving cursor into the array
 - if NO_DATA_FOUND, this means there were fewer than g_l_restart_max_counter records fetched or none were fetched. Set a flag to eventually break from the loop.
 - Calculate the num_records_to_process in the array by subtracting previously processed records from NUM_RECORDS_PROCESSED (this is an automatically kept variable that keeps how many records have been fetched from a cursor).
 - If restart_string is NULL (happens on a fresh start only)
 - 1) Populate it with the first LUW information
- For 0 to < num_records_to_process, loop
 - If LUW has NOT changed
 - 2) If the 3rd byte of code is 'C' or 'B', write_array() for cost
 - 3) If the 3rd byte of code is 'R' or 'B', write_array() for retail
 - ELSE
 - 4) Write_to_table() to fif_gl_data
 - 5) Call restart_commit()
 - 6) Set restart_string to new LUW
 - 7) If the 3rd byte of code is 'C' or 'B', write_array() for cost
 - 8) If the 3rd byte of code is 'R' or 'B', write_array() for retail
 - Add num_records_to_process to the previously processed records
- If loop breaking flag is set
 - break
 - END WHILE
 - Call write_to_table() -- this will insert the final LUW from the insert_array

final() -

- Close off restart recovery processing

size_arrays() -

- Allocates the memory for the two array structures.

write_array(array_index, cost_retail_flag) -

- write cost/retail depending on flag passed in

- increment array_size variable
- if array_size = g_l_restart_max then call write_to_table()
write_to_table() -
- For insert_array_size records, an array insert into fif_gl_data
- Set insert_array_size counter = 0.
- If there are insertion errors, the program should return fatally.

Functional Level Description

All required stock ledger data must be processed to the GL interface table within Retek. Errors will be handled in the normal batch method.

I/O Specification

None.

Technical Issues

None.

[Fifglndn2107.pc]

Design Overview

This process will summarize stock ledger data from the transaction staging table `if_tran_data`, based on the level of information required, and post it to a staging table within Retek for transfer to a financial application's general ledger.

This process reads from the `if_tran_data` table, instead of the actual `tran_data` table itself. Hence, it requires the 'PRE-SALDLY' and 'PRE-FIFGLDN1' run of `prepost.pc` to have occurred prior to its initiation. It can run in parallel to `fifglnd1.pc`.

New Codes Table Information

The new code type that will be created for this process is the General Ledger – Rolled Transaction 'GLRT' type. This will include transaction types/amount types to go through the roll-up processing if the feed to GL is to be daily. The code for this will be a 3-character field, the first two characters determining the transaction code, the third character determining if the cost or retail component of the amount is to be bridged ('C' for Cost, 'R' for Retail, and 'B' for Both). For instance, if the code 01R is on the table, the net sales retail will be bridged at the sku-detail level.

Note: There are several risks in the maintenance of these codes types:

The codes table is not masked, and one could incorrectly enter an R12 instead of 12R, etc. This will result in not sending the code 12 GL transactions.

'12R' could be entered in more than one codes type. This will result in duplicate processing, although at different levels of roll-up. There is no cross-checking to verify that a code entered in one code type is not in the other.

Daily-level GL Feed

The new program `fifglnd2.pc` will read and summarize the `if_tran_data` table for all transaction types/amount types in the codes table, and post it to a Retek general ledger staging table (`fif_gl_data`) at the department, class, or subclass level.

The new program `fifglnd2.pc` can run in parallel to the `saldly` process, as well as the `fifglnd1.pc` process, but must run after the `if_tran_data` load process, as it will process rows on `if_tran_data`.

The local currency will be retrieved for all locations from their corresponding store/warehouse rows.

Scheduling Constraints

Processing Cycle Daily, possibly in parallel to saldly, after
PREPOST('pre-saldly' and 'pre-fifgldn1').

Scheduling Diagram:

Pre-Processing:

Post-Processing

Threading Scheme: store/wh

Restart Recovery

Driving Cursor

This program will be fully restartable. It will use three different driving cursors, based on the input parameter (DEPT, CLASS, SUBCLASS):

If 'DEPT' is the input parameter, the following cursor will be used:

```
Select td.Dept,
      -1,
      -1,
      td.Store,
      td.Wh,
      td.tran_date,
      Td.Tran_code,
      sum(Td.Total_cost),
      sum(Td.Total_retail),
      cd.code,
      '||to_char(td.dept)
from if_tran_data td,
      code_detail cd,
      v_restart_store_wh v
where (cd.code_type = 'GLRT')
```

```
and (substr(cd.code,1,2)=td.tran_code)

and v.driver_name = :os_restart_driver_name

and v.driver_value = :td.store

and v.num_threads = :oi_restart_num_threads

and v.thread_val = :oi_restart_thread_val

and td.dept > NVL(:ora_restart_dept, td.dept-1)

group by td.store,

        td.wh,

        td.dept,

        td.tran_code,

        td.tran_date,

        cd.code

order by td.dept
```

If 'CLASS' is the input parameter, the following cursor will be used:

```
Select td.Dept,

        Td.class,

        -1,

        td.Store,

        td.Wh,

        td.tran_date,

        Td.Tran_code,

        sum(Td.Total_cost),

        sum(Td.Total_retail),

        cd.code,

        '||to_char(td.dept)||';||to_char(td.class)

from if_tran_data td,
```



```

        code_detail cd,
        v_restart_store_wh v
where (cd.code_type = 'GLRT')
      and (substr(cd.code,1,2)=td.tran_code)
      and v.driver_name = :os_restart_driver_name
      and v.driver_value = :td.store
      and v.num_threads = :oi_restart_num_threads
      and v.thread_val = :oi_restart_thread_val
      and (td.dept > NVL(:ora_restart_dept, td.dept-1) OR
           (td.dept = :ora_restart_dept AND
            (td.class > :ora_restart_class)))
group by td.store,
         td.wh,
         td.dept,
         td.class,
         td.tran_code,
         td.tran_date,
         cd.code
order by td.dept,
         td.class

```

If 'SUBCLASS' is the input parameter, the following cursor will be used:

```

Select td.Dept,
       Td.class,
       Td.subclass,
       td.Store,
       td.Wh,

```

```
        td.tran_date,
        Td.Tran_code,
        sum(Td.Total_cost),
        sum(Td.Total_retail),
        cd.code,
        '||to_char(td.dept)||';'||to_char(td.class)||';'||to_char(td.subclass)
from if_tran_data td,
        code_detail cd,
        v_restart_store_wh v
where (cd.code_type = 'GLRT')
      and (substr(cd.code,1,2)=td.tran_code)
      and v.driver_name = :os_restart_driver_name
      and v.driver_value = :td.store
      and v.num_threads = :oi_restart_num_threads
      and v.thread_val = :oi_restart_thread_val
      and (td.dept > NVL(:ora_restart_dept, td.dept-1) OR
           (td.dept = :ora_restart_dept AND
            (td.class > :ora_restart_class OR
             (td.class = :ora_restart_class AND
              (td.subclass > :ora_restart_subclass))))))
group by td.store,
        td.wh,
        td.dept,
        td.class,
        td.subclass,
        td.tran_code,
```

```

        td.tran_date,

        cd.code

order by td.dept,

        td.class,

        td.subclass

```

Program Flow

The high-level processing for this program is summarized in the following 3 steps:

1. Read the row
 2. Obtain its cost component from code_detail (substr(code, 3,1)). If it is 'C' for Cost, insert a row to fif_gl_data for the cost amount. If it is 'R', insert a row for the retail amount. If it is 'B' for both, insert one row for each.
 3. If the processing/insert was successful, continue, else write a row to the if_errors interface error table.
- This program will take the value of system_options.gl_rollup. There will be a value of 'D' for department, 'C' for class, or 'S' for subclass.
 - To facilitate the dept/class/subclass nature of this program, three integer global bind vars will be declared: Dept_rollup, class_rollup, and subclass_rollup. Depending on the system_options value described above, the appropriate variable will be set to 1 and the others to 0 (zero). If/then calls such as "if (dept_rollup) { <statement> };" will allow for easy readability of the code.
 - In addition to the insert array structure (with fields mirroring the fif_gl_data table structure), one structure (mirroring the subclass cursor field list plus a field for the LUW info retrieved from the cursor) should be declared. Depending on the level of rollup desired, the class or class & subclass fields will be populated with '-1', but this method is cleaner than having 3 structures to track.

init() -

- Initialize restart recovery.
- Set up array structures for the driving cursor fetch as well as one for inserts into the FIF_GL_DATA table (dept, class, subclass, location, loc_type, tran_date, tran_code, adj_code, amount, cost_retail_flag, ref_no_1, ref_no_2, sku, currency_code in the structure).
- Normal array sizing should occur [size_arrays()]

- Get value from system_options (as described above). If it is a department level rollup, initialize only the ora_restart_dept var. If it's class, initialize only ora_restart_dept and ora_restart_class, etc.

process() -

- open the appropriate driving cursor
- WHILE(1)
 - fetch g_l_restart_max_counter records from appropriate driving cursor into the array
 - if NO_DATA_FOUND, this means there were fewer than g_l_restart_max_counter records fetched or none were fetched. Set a flag to eventually break from the loop.
 - Calculate the num_records_to_process in the array by subtracting previously processed records from NUM_RECORDS_PROCESSED (this is an automatically kept variable that keeps how many records have been fetched from a cursor).
 - If restart_string is NULL (happens on a fresh start only)
 - 9) Populate it with the first LUW information
 - For 0 to < num_records_to_process, loop
 - If LUW has NOT changed
 - 10) If the 3rd byte of code is 'C' or 'B', write_array() for cost
 - 11) If the 3rd byte of code is 'R' or 'B', write_array() for retail
 - ELSE
 - 12) Write_to_table() to fif_gl_data
 - 13) Call restart_commit()
 - 14) Set restart_string to new LUW
 - 15) If the 3rd byte of code is 'C' or 'B', write_array() for cost
 - 16) If the 3rd byte of code is 'R' or 'B', write_array() for retail
 - 17) Add num_records_to_process to the previously processed records
 - If loop breaking flag is set
 - 18) break
 - 19) END WHILE
 - 20) Call write_to_table() -- this will insert the final LUW from the insert_array

size_arrays() -

- Allocates the memory for the two array structures. (the fetch array and the insert array)

write_array(array_index, cost_retail_flag) -

- Fetch currency code from store or wh table.
- write cost/retail depending on flag passed in
- increment array_size variable
- if array_size = g_l_restart_max then call write_to_table()

write_to_table() -

- For insert_array_size records, an array insert into fif_gl_data. If -1's are in class and/or subclass, insert them as normal.
- Set insert_array_size counter = 0.
- If there are insertion errors, the program should return fatally.

final() -

- Close off restart recovery processing

Functional Level Description

All required stock ledger data must be processed to the GL interface table within Retek. Errors will be handled using the normal batch techniques.

I/O Specification

None.

Technical Issues

None.

[Fifgldn3107.pc]

Design Overview

This process will summarize stock ledger data from the monthly stock ledger table `month_data`, based on the level of information required, and post it to a staging table within Retek for transfer to a financial application's general ledger.

Since this process reads from the `month_data` table, it requires the sales monthly process SALMTH to have run. It can run any time after the close of the last day of the month.

New Codes Table Information

The new code type that will be created for this process is the General Ledger – Monthly Transaction 'GLMT' type. This will include transaction types/amount types to go through the roll-up processing if the feed to GL is to be monthly. The code for this will be a 3-character field, the first two characters determining the transaction code, the third character determining if the cost or retail component of the amount is to be bridged ('C' for Cost, 'R' for Retail, and 'B' for Both). For instance, if the code 01R is on the table, the net sales retail will be bridged at the department, class, or subclass level.

Since the monthly amounts are no longer maintained as transaction codes, this process will need to map certain amounts back to `tran_codes`. Some amounts are calculated from existing amount fields, and are not present on the `tran_data` level. These amounts will be given `tran_codes` specifically for bridging to a general ledger. The following list is the field mapping for the process:

Month_data field	Tran code	Cost/ Retail	Comment
OPN_STK_RETAIL	50	R	Monthly Calculated Figure
OPN_STK_COST	50	C	Monthly Calculated Figure
STOCK_ADJ_RETAIL	22	R	
STOCK_ADJ_COST	22	C	
PURCH_RETAIL	20	R	
PURCH_COST	20	C	
RTV_RETAIL	24	R	
RTV_COST	24	C	
FREIGHT_COST	26	C	
TSF_IN_RETAIL	30	R	
TSF_IN_COST	30	C	

Month_data field	Tran code	Cost/ Retail	Comment
TSF_OUT_RETAIL	32	R	
TSF_OUT_COST	32	C	
NET_SALES_RETAIL	01	R	
NET_SALES_RETAIL_EX_VAT	02	R	
NET_SALES_COST	01	C	
RETURNS_RETAIL	04	R	
RETURNS_COST	04	C	
MARKUP_RETAIL	11	R	
MARKUP_CAN_RETAIL	12	R	
CLEAR_MARKDOWN_RETAIL	16	R	
PERM_MARKDOWN_RETAIL	13	R	
PROM_MARKDOWN_RETAIL	15	R	
MARKDOWN_CAN_RETAIL	14	R	
SHRINKAGE_COST	51	C	Monthly Calculated Figure
SHRINKAGE_RETAIL	51	R	Monthly Calculated Figure
EMPL_DISC_RETAIL	60	R	
WORKROOM_AMT	80	R	
CASH_DISC_AMT	81	R	
CLS_STK_RETAIL	52	R	Monthly Calculated Figure
CLS_STK_COST	52	C	Monthly Calculated Figure
GROSS_MARGIN_AMT	53	R	Monthly Calculated Figure
COST_VARIANCE_AMT	70	C	
HTD_GAFS_RETAIL	54	R	Monthly Calculated Figure
HTD_GAFS_COST	54	C	Monthly Calculated Figure
INTER_STOCKTAKE_SALES_AMT	55	C	Monthly Calculated Figure
INTER_STOCKTAKE_SHRINK_AMT	56	C	Monthly Calculated Figure
STOCKTAKE_MTD_SALES_AMT	57	C	Monthly Calculated Figure
STOCKTAKE_MTD_SHRINK_AMT	58	C	Monthly Calculated Figure
STOCKTAKE_BOOKSTK_RETAIL	59	R	Monthly Calculated Figure
STOCKTAKE_BOOKSTK_COST	59	C	Monthly Calculated Figure
STOCKTAKE_ACTSTK_RETAIL	61	R	Monthly Calculated Figure

Month_data field	Tran code	Cost/ Retail	Comment
STOCKTAKE_ACTSTK_COST	61	C	Monthly Calculated Figure
RECLASS_IN_COST	34	C	
RECLASS_IN_RETAIL	34	R	
RECLASS_OUT_COST	36	C	
RECLASS_OUT_RETAIL	36	R	

Note: There are several risks in the maintenance of these codes types:

The codes table is not masked, and one could incorrectly enter an R12 instead of 12R, etc. This will result in not sending the code 12 GL transactions.

‘12R’ could be entered in more than one codes type. This will result in duplicate processing, although at different levels of roll-up or different processing times (daily versus monthly). There is no cross-checking to verify that a code entered in one code type is not in the other.

Monthly-level GL Feed

New processes:

The new program fifgldn3.pc will read and summarize the month_data table for all transaction types/amount types in the codes table, and post it to a Retek general ledger staging table (fif_gl_data) at the department, class, or subclass level.

The new program fifgldn3.pc must run after the salmth process, as it will process rows on month_data.

The local currency will be retrieved for all locations from their corresponding store/warehouse rows.

Scheduling Constraints

Processing Cycle: Monthly, after salmth and sysdte have run.

Restart Recovery

Driving Cursor

There will be one cursor for DEPT-level roll-up:

```
select md.dept,  
       -1,  
       -1,  
       md.store,  
       md.wh,  
       NVL(decode(cd.code, '50R',sum(md.opn_stk_retail),  
                  '50C',sum(md.opn_stk_cost),  
                  '22R',sum(md.stock_adj_retail),  
                  '22C',sum(md.stock_adj_cost),  
                  '20R',sum(md.purch_retail),  
                  '20C',sum(md.purch_cost),  
                  '24R',sum(md.rtv_retail),  
                  '24C',sum(md.rtv_cost),  
                  '26C',sum(md.freight_cost),  
                  '30R',sum(md.tsf_in_retail),  
                  '30C',sum(md.tsf_in_cost),  
                  '32R',sum(md.tsf_out_retail),  
                  '32C',sum(md.tsf_out_cost),  
                  '01R',sum(md.net_sales_retail),  
                  '02R',sum(md.net_sales_retail_ex_vat),  
                  '01C',sum(md.net_sales_cost),  
                  '04R',sum(md.returns_retail),  
                  '04C',sum(md.returns_cost),
```

'11R',sum(md.markup_retail),
'12R',sum(md.markup_can_retail),
'16R',sum(md.clear_markdown_retail),
'13R',sum(md.perm_markdown_retail),
'15R',sum(md.prom_markdown_retail),
'14R',sum(md.markdown_can_retail),
'51C',sum(md.shrinkage_cost),
'51R',sum(md.shrinkage_retail),
'60R',sum(md.empl_disc_retail),
'80R',sum(md.workroom_amt),
'81R',sum(md.cash_disc_amt),
'52R',sum(md.cls_stk_retail),
'52C',sum(md.cls_stk_cost),
'53R',sum(md.gross_margin_amt),
'70C',sum(md.cost_variance_amt),
'54R',sum(md.htd_gafs_retail),
'54C',sum(md.htd_gafs_cost),
'55C',sum(md.inter_stocktake_sales_amt),
'56C',sum(md.inter_stocktake_shrink_amt),
'57C',sum(md.stocktake_mtd_sales_amt),
'58C',sum(md.stocktake_mtd_shrink_amt),
'59R',sum(md.stocktake_bookstk_retail),
'59C',sum(md.stocktake_bookstk_cost),
'61R',sum(md.stocktake_actstk_retail),
'61C',sum(md.stocktake_actstk_cost),
'34C',sum(md.reclass_in_cost),

```

        '34R',sum(md.reclass_in_retail),
        '36C',sum(md.reclass_out_cost),
        '36R',sum(md.reclass_out_retail),
        0),0),
    cd.code,
    '||to_char(md.dept)
from month_data md,
    v_restart_store_wh v,
    code_detail cd
where cd.code_type = 'GLMT'
    and md.half_no = :last_half_no
    and md.month_no = :last_month_no
    and md.currency_ind = 'L'
    and v.driver_name = :os_restart_driver_name
    and v.driver_value = md.store
    and v.num_threads = :oi_restart_num_threads
    and v.thread_val = :oi_restart_thread_val
    and md.dept > NVL(:ora_restart_dept, md.dept - 1)
group by md.store,
    md.wh,
    md.dept,
    cd.code
order by md.dept

```

Also, one for CLASS-level roll-up:

```

SELECT md.dept,
    md.class,

```

```
-1,  
  
md.store,  
  
md.wh,  
  
NVL(decode(cd.code, '50R',sum(md.opn_stk_retail),  
        '50C',sum(md.opn_stk_cost),  
        '22R',sum(md.stock_adj_retail),  
        '22C',sum(md.stock_adj_cost),  
        '20R',sum(md.purch_retail),  
        '20C',sum(md.purch_cost),  
        '24R',sum(md.rtv_retail),  
        '24C',sum(md.rtv_cost),  
        '26C',sum(md.freight_cost),  
        '30R',sum(md.tsf_in_retail),  
        '30C',sum(md.tsf_in_cost),  
        '32R',sum(md.tsf_out_retail),  
        '32C',sum(md.tsf_out_cost),  
        '01R',sum(md.net_sales_retail),  
        '02R',sum(md.net_sales_retail_ex_vat),  
        '01C',sum(md.net_sales_cost),  
        '04R',sum(md.returns_retail),  
        '04C',sum(md.returns_cost),  
        '11R',sum(md.markup_retail),  
        '12R',sum(md.markup_can_retail),  
        '16R',sum(md.clear_markdown_retail),  
        '13R',sum(md.perm_markdown_retail),  
        '15R',sum(md.prom_markdown_retail),
```

```
'14R',sum(md.markdown_can_retail),  
'51C',sum(md.shrinkage_cost),  
'51R',sum(md.shrinkage_retail),  
'60R',sum(md.empl_disc_retail),  
'80R',sum(md.workroom_amt),  
'81R',sum(md.cash_disc_amt),  
'52R',sum(md.cls_stk_retail),  
'52C',sum(md.cls_stk_cost),  
'53R',sum(md.gross_margin_amt),  
'70C',sum(md.cost_variance_amt),  
'54R',sum(md.htd_gafs_retail),  
'54C',sum(md.htd_gafs_cost),  
'55C',sum(md.inter_stocktake_sales_amt),  
'56C',sum(md.inter_stocktake_shrink_amt),  
'57C',sum(md.stocktake_mtd_sales_amt),  
'58C',sum(md.stocktake_mtd_shrink_amt),  
'59R',sum(md.stocktake_bookstk_retail),  
'59C',sum(md.stocktake_bookstk_cost),  
'61R',sum(md.stocktake_actstk_retail),  
'61C',sum(md.stocktake_actstk_cost),  
'34C',sum(md.reclass_in_cost),  
'34R',sum(md.reclass_in_retail),  
'36C',sum(md.reclass_out_cost),  
'36R',sum(md.reclass_out_retail),  
0),0),
```

```
cd.code,
```

```
        '||to_char(md.dept)||'||to_char(md.class)

FROM month_data md,

        v_restart_store_wh v,

        code_detail cd

WHERE cd.code_type = 'GLMT'

        AND md.half_no = :last_half_no

        AND md.month_no = :last_month_no

        AND md.currency_ind = 'L'

        AND v.driver_name = :os_restart_driver_name

        AND v.driver_value = md.store

        AND v.num_threads = :oi_restart_num_threads

        AND v.thread_val = :oi_restart_thread_val

        AND (md.dept > NVL(:ora_restart_dept, md.dept - 1) OR

                (md.dept = :ora_restart_dept and

                 (md.class > :ora_restart_class)))

GROUP BY md.store,

        md.wh,

        md.dept,

        md.class,

        cd.code

ORDER BY md.dept,

        md.class
```

Lastly, there will be one for SUBCLASS-level roll-up:

```
SELECT md.dept,

        md.class,

        md.subclass,
```

```
md.store,
md.wh,
NVL(decode(cd.code, '50R',sum(md.opn_stk_retail),
      '50C',sum(md.opn_stk_cost),
      '22R',sum(md.stock_adj_retail),
      '22C',sum(md.stock_adj_cost),
      '20R',sum(md.purch_retail),
      '20C',sum(md.purch_cost),
      '24R',sum(md.rtv_retail),
      '24C',sum(md.rtv_cost),
      '26C',sum(md.freight_cost),
      '30R',sum(md.tsf_in_retail),
      '30C',sum(md.tsf_in_cost),
      '32R',sum(md.tsf_out_retail),
      '32C',sum(md.tsf_out_cost),
      '01R',sum(md.net_sales_retail),
      '02R',sum(md.net_sales_retail_ex_vat),
      '01C',sum(md.net_sales_cost),
      '04R',sum(md.returns_retail),
      '04C',sum(md.returns_cost),
      '11R',sum(md.markup_retail),
      '12R',sum(md.markup_can_retail),
      '16R',sum(md.clear_markdown_retail),
      '13R',sum(md.perm_markdown_retail),
      '15R',sum(md.prom_markdown_retail),
      '14R',sum(md.markdown_can_retail),
```

```
'51C',sum(md.shrinkage_cost),
'51R',sum(md.shrinkage_retail),
'60R',sum(md.empl_disc_retail),
'80R',sum(md.workroom_amt),
'81R',sum(md.cash_disc_amt),
'52R',sum(md.cls_stk_retail),
'52C',sum(md.cls_stk_cost),
'53R',sum(md.gross_margin_amt),
'70C',sum(md.cost_variance_amt),
'54R',sum(md.htd_gafs_retail),
'54C',sum(md.htd_gafs_cost),
'55C',sum(md.inter_stocktake_sales_amt),
'56C',sum(md.inter_stocktake_shrink_amt),
'57C',sum(md.stocktake_mtd_sales_amt),
'58C',sum(md.stocktake_mtd_shrink_amt),
'59R',sum(md.stocktake_bookstk_retail),
'59C',sum(md.stocktake_bookstk_cost),
'61R',sum(md.stocktake_actstk_retail),
'61C',sum(md.stocktake_actstk_cost),
'34C',sum(md.reclass_in_cost),
'34R',sum(md.reclass_in_retail),
'36C',sum(md.reclass_out_cost),
'36R',sum(md.reclass_out_retail),
0),0),
cd.code,
'||to_char(md.dept)||'||to_char(md.class)||'||to_char(md.subclass)
```



```
FROM month_data md,
     v_restart_store_wh v,
     code_detail cd
WHERE cd.code_type = 'GLMT'
      AND md.half_no = :last_half_no
      AND md.month_no = :last_month_no
      AND md.currency_ind = 'L'
      AND v.driver_name = :os_restart_driver_name
      AND v.driver_value = md.store
      AND v.num_threads = :oi_restart_num_threads
      AND v.thread_val = :oi_restart_thread_val
      AND (md.dept > NVL(:ora_restart_dept, md.dept - 1) OR
           (md.dept = :ora_restart_dept and
            (md.class > :ora_restart_class or
             (md.class = :ora_restart_class and
              (md.subclass > :ora_restart_subclass)))))
GROUP BY md.store,
         md.wh,
         md.dept,
         md.class,
         md.subclass,
         cd.code
ORDER BY md.dept,
         md.class,
         md.subclass
```

Program Flow

- This program will take the value of `system_options.gl_rollup`. There will be a value of 'D' for department, 'C' for class, or 'S' for subclass.
- To facilitate the dept/class/subclass nature of this program, three integer global bind vars will be declared: `Dept_rollup`, `class_rollup`, and `subclass_rollup`. Depending on the `system_options` value described above, the appropriate variable will be set to 1 and the others to 0 (zero). If/then calls such as "if (`dept_rollup`) { <statement> };" will allow for easy readability of the code.
- In addition to the insert array structure (with fields mirroring the `fif_gl_data` table structure), one structure (mirroring the **subclass** cursor field list) should be declared. Depending on the level of rollup desired, the class or class & subclass fields will be populated with '-1', but this method is cleaner than having 3 structures to track.
- `Fif_gl_data.tran_date` will always be populated with `system_variables.last_eom_date`

`init()` -

- Initialize restart recovery.
- Set up array structures for the driving cursor fetch as well as one for inserts into the `FIF_GL_DATA` table (`dept`, `class`, `subclass`, `location`, `loc_type`, `tran_date`, `tran_code`, `adj_code`, `amount`, `cost_retail_flag`, `ref_no_1`, `ref_no_2`, `sku`, `currency_code` in the structure).
- Normal array sizing should occur [`size_arrays()`]
- Get value from `system_options` (as described above). If it is a department level rollup, initialize only the `ora_restart_dept` var. If it's class, initialize only `ora_restart_dept` and `ora_restart_class`, etc.
- Get values from `system_vars`: `last_eom_half_no`, `last_eom_month_no`, & `last_eom_date`

`process()` -

- open the appropriate driving cursor
- `WHILE(1)`
 - fetch `g_l_restart_max_counter` records from appropriate driving cursor into the array
 - if `NO_DATA_FOUND`, this means there were fewer than `g_l_restart_max_counter` records fetched or none were fetched. Set a flag to eventually break from the loop.

- Calculate the num_records_to_process in the array by subtracting previously processed records from NUM_RECORDS_PROCESSED (this is an automatically kept variable that keeps how many records have been fetched from a cursor).
 - 21) If restart_string is NULL (happens on a fresh start only)
 - Populate it with the first LUW information
 - For 0 to < num_records_to_process, loop
 - If LUW has NOT changed
 - 22) fill_array()
 - ELSE
 - 23) Write_array() to fif_gl_data
 - 24) Call restart_commit()
 - 25) Set restart_string to new LUW
 - 26) fill_array() for cost
 - Add num_records_to_process to the previously processed records
 - If loop breaking flag is set
 - break
 - END WHILE
 - Call write_array() -- this will insert the final LUW from the insert_array
- fill_array(array_index, cost_retail_flag) -
- Fetch currency code from store or wh table.
 - write cost/retail depending on flag passed in
 - increment array_size variable
 - if array_size = g_l_restart_max then call resize_array()
- write_array() -
- For insert_array_size records, an array insert into fif_gl_data. If -1's are in class and/or subclass, insert them as normal.
 - Set insert_array_size counter = 0.
 - If there are insertion errors, the program should return fatally.

size_arrays() -

- Allocates the memory for the two array structures. (the fetch array and the insert array)

resize_array() -

- increase the size of the insert array by another g_l_restart_max slots

final() -

- Close off restart recovery processing

Functional Level Description

All required stock ledger data must be processed to the GL interface table within Retek. Errors with the download will require an abort.

I/O Specification

None.

Technical Issues

None.

[fifinvcu107.pc]

Design Overview

This batch program will read records from the Retek staging table IIF_HEAD that have a payment status of 'P' (ready to be paid —these are invoices that are already matched). The program will use these invoices to populate the Oracle AP_EXPENSE_REPORT_HEADERS_ALL table. The associated records in the IIF_DETAIL, IIF_NON_MERCH and IIF_MERCH_VAT tables will be used to populate the Oracle AP_EXPENSE_REPORT_LINES_ALL table. The invoice amount for each line record is VAT-inclusive, so that the total invoice amount (sum of the line amount) in the header level is VAT-inclusive. Each selected invoice that has header and detail records successfully written to the Oracle interface table will then be deleted immediately from Retek's staging tables. If an error occurs in the insert to the Oracle tables, a record will be written to the error table (if_errors), and records will not be deleted from Retek's staging tables.

Tables used:

Table	Select	Insert	Update	Delete
iif_head	Yes	No	No	Yes
iif_detail	Yes	No	No	Yes
iif_merch_vat	Yes	No	No	Yes
iif_match_detail	Yes	No	No	Yes
iif_non_merch	Yes	No	No	Yes
po_vendors_	Yes	No	No	No
vat_code_rates	Yes	No	No	No
if_errors	No	Yes	No	No
ap_expense_report_headers_all	No	Yes	No	Yes
ap_expense_report_lines_all	No	Yes	No	No
period	Yes	No	No	No
system_options	Yes	No	No	No
shipment	Yes	No	No	No
shipsku	Yes	No	No	No
addr	Yes	No	No	No
ordloc	Yes	No	No	No
fif_ora_cross_ref	Yes	No	No	No
fif_ora_gl_setup	Yes	No	No	No

Table	Select	Insert	Update	Delete
po_vendor_sites_all	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Daily, Phase II

Scheduling Diagram: Run after invcpost.pc, which populates the iif_* tables.

Pre-Processing: invcpost.pc

Threading Scheme: not threaded

Restart Recovery

Driving Cursor:

```
SELECT ih.invc_id,  
       DECODE(ih.invc_type,'T','STANDARD',  
              'N','STANDARD',  
              'O','STANDARD',  
              'M','CHECKME',  
              'SWITCH'),  
       ih.supplier,  
       ih.partner_type,  
       ih.partner_id,  
       ih.direct_ind,  
       NVL(ih.ext_ref_no,(NVL(ih.ref_rtv_order_no,'-999'))),  
       addr.oracle_vendor_site_id,  
       ih.payment_ref_no,  
       ih.proof_of_delivery_no,  
       TO_CHAR(ih.invc_date,'YYYYMMDD'),  
       ih.currency_code,  
       ih.exchange_rate*:pd_ten10,
```

```

        ih.total_payment_merch_cost,
        NVL(SUBSTR(ih.comments,1,240),' '))
FROM iif_head ih,
        addr
WHERE ih.payment_status='P'
        AND addr.addr_key = ih.addr_key
        AND ih.invc_id > NVL(:ps_restart_invc_id,-999)
        ORDER BY ih.invc_id;

```

The logical unit of work will be the invoice.

The commit_max_ctr field should be set to prevent excessive rollback space usage. The recommended commit counter setting is 1000 records (subject to change based on experimentation).

Function Level Description

A Note Regarding Invoice Type and CCIDs:

The Oracle invoice type will be 'STANDARD' for all Retek type 'I' (regular), 'N' (non-merchandise), and 'O' (consignment) invoices. If the invoice is Retek type 'M' (credit memo), the invoice type will be 'STANDARD' if the total cost on iif_head is greater than or equal to zero, and 'CREDIT' if the total cost is less than zero.

For Retek invoice types 'C'redit and 'D'ebit memo, the following behavior is required:

1. If the total cost on iif_head is greater than or equal to zero, or if there is no total cost, the Oracle invoice type will be 'CREDIT'.
2. If the total cost is less than zero, the Oracle invoice type will be 'STANDARD'.
3. The end result total amount (created from the detail level sums) must have its sign switched. This should be done as follows: (The multiplication will be done using a variable that is 1 unless a switch is needed, at which point it will be set to -1. The multiplication will always be done, but will be invisible in the result unless a switch must take place.)
 - For iif_non_merch and iif_merch_vat, multiply the cost or amount (whichever you get—they will both be placed in the amount field of ap_expense_report_lines_all) by -1

- For iif_detail, look at the quantity and cost:
 - If both are ≥ 0 , multiply the quantity by -1 .
 - If only one is ≥ 0 , multiply that one by -1 .
 - If both are negative, multiply both by -1 .

BOTH standard and credit invoices must get ccid's (which are inserted into the code_combination_id field of the ap_expense_report_lines_all table). Standard invoices should get the dr_ccid from fif_ora_cross_ref; credit invoices should get the cr_ccid.

If the invoice requires a ccid but has no item detail records (header only), the ccid will be defaulted to the value on fif_ora_cross_ref, where dept=-999, class=-1, subclass=-1, location=-1, tran_code=21, cost or retail flag='C', and line_type = 'M'.

If the invoice has item detail records, the invoice will be rejected if a ccid cannot be found for all of the item detail records. An error indicating the dept/class/subclass/location for which a ccid could not be found will be written to the if_errors table. This is done so the user can enter the required information online before re-running the program (using the GL cross reference form in the Financials folder of RMS).

Init()

- Initialize restart recovery.
- Retrieve system date from PERIOD, financial_ap and vat indicator from SYSTEM_OPTIONS, and last_update_id, org_id, and set_of_books_id from FIF_ORA_GL_SETUP. If the financial_ap is not 'O'racle, terminate.

Process()

Call size_head_array and size_detail_array to allocate memory for arrays.

- Open the driving cursor in a loop. Fetch the data into an array.
- For each invoice fetched.
- Call get_oracle_invoice to create an Oracle invoice ID.
- Call get_vendor_info to get the vendor_id and accts_pay_code_combination_id.
- If it is a credit memo (type 'M') and the total amount is positive, change the invoice type to 'STANDARD'. Otherwise, change it to 'CREDIT'. If it is type 'C' or 'D', change the invoice type (see the rules in the note above) and set a multiply flag to -1 .

- Call `get_ccid` to get the ‘miscellaneous’ CCID that will be used by vat and non_merch lines (or the dummy detail line if there is only header information).
- Call `get_details` to get detail level information.
- Call `insert_ap_expense_header` to insert header-level information. If it fails, write a message to the `if_errors` table.
- If the insert was successful, call `insert_detail` to insert detail-level information. If `insert_detail` fails, call `delete_head` to delete the `ap_expense_report_headers_all` record just inserted, and write a message to `if_errors`.
- If there are no detail records to insert, call `make_dummy_detail` to insert a dummy detail record (as is required by Oracle).
- If all inserts for this invoice were successful, delete this invoice from `lif_head`, `lif_detail`, `lif_non_merch`, `lif_match_detail`, and `lif_merch_vat` tables by calling `delete_iif`.

`get_ccid`

Fetch `dr_ccid` or `cr_ccid` as appropriate from the `fif_ora_cross_ref` table for a given invoice type (standard or credit) dept, class, subclass, and location, and a `tran_code` of 21 (the invoice tran code).

(Use -1 for a missing location.) and `line_type`.

Verify that the obtained `ccid` exists in Oracle view `gl_code_combinations_cai_v` and `gl_code_combinations_cai_v.detail_posting_allowed_flag` is ‘Y’ and `gl_code_combinations_cai_v.enabled_flag` is ‘Y’.

`get_details`

Call `get_iif_detail`, `get_non_merch`, and `get_merch_vat` to get detail-level information for the invoice.

`get_iif_detail`

Retrieve records from the iif_detail table for a given invoice or future insert into the ap_expense_report_headers_all table. It will first get SKU information from iif_detail and desc_look. If VAT is turned on, it will calculate the VAT-inclusive cost and will find the VAT code for the given vat rate. The cost*qty will be saved to go into the detail-level amount field, and will be added to the total invoice cost. The iif_match_detail and ordloc tables will then be searched for a corresponding order, shipment, and order location for the given invoice and SKU. If there are multiple orders associated with a SKU, only the first one will be fetched. Get_ccid will then be called. If a ccid is not found, write an error message to the if_errors table, giving the dept, class, subclass, and location information so that a user can update the account information as needed. (Loop through all iif_detail records; don't stop on the first one that a ccid is not found for. Don't do any further processing if the ccid is missing.)

get_non_merch:

Retrieve the non_merch_amt and vat_code from the lif_non_merch table. Add those records to the detail array, get CCID for each non_merch_code type.

Open a loop to retrieve each of the detail records on the lif_non_merch table,

- Fetch cursor C_invoice_non_merch. If no data is found, exit from the loop.
- Increment sequence number $sequence = sequence + 1$.
- If vat_code is not NULL: get vat rate from vat_code_rates. Calculate the invoice non merch amount, using the following formula: $non\_merch\_amt + non\_merch\_amt * vat\_rate/100$.
- Set the amount = non_merch_amount.

Prorate_non_merch:

Prorate non_merch (Freight, Tax, and Miscellaneous) amount by totaling up the cost of each item and then determine the percent of that total for each item. Apply that same percent to the (as exemple) freight total. Distribute the freight to each line items. *An example of prorates Non_Merch line type is appended in the last part of this document.

Build_inv_detail:

When we do summary matching, if it is matched we only get a header record but no detail. We need to find detail records for this invoice so that Ap_expense_report_lines_all can be populated and the correct ccid will be retrieved for the matched invoice. This can be accomplished by getting the relation from Invoice id with shipment and Order_no in the invc_xref table. For the shipment we will be able to find shu information from shipsku table. Thus the sku information was extracted and used to build invoice detail lines for populating Ap-expense_report_lines_all.

get_merch_vat:

Retrieve the total_cost_excl_vat and the vat_code from the iif_merch_vat table. Add those records to the detail array, using the 'miscellaneous' CCID.

Open a loop to retrieve each of the detail records on the Iif_merch_vat table,

- Fetch cursor C_invoice_merch_vat. If no data is found, exit from the loop.
- Increment sequence number $sequence = sequence + 1$.
- Get vat rate from vat_code_rates. Calculate invoice merch vat amount, using the following formula: $total\_cost\_excl\_vat + total\_cost\_excl\_vat * vat\_rate/100$.

Insert_ap_invoice

This function inserts a record to the Ap_expense_report_headers_all table. The source field should be 'RETEK' (see below for exact details of what goes in what field)

Insert_detail

Do an array insert of the detail array to the Ap_expense_report_lines_all table (see table below for exact details of what goes in what field).

delete_iif

For a given invoice ID, delete from the iif_non_merch, iif_match_detail, iif_merch_vat, iif_detail, and iif_head tables.

write_error_table

Insert a given unit of work and error message into the if_errors table.

Size_head_array

Allocate memory for the header array (used for array fetches from the driving cursor).

Size_detail_array:

Allocate memory for the detail array (used for array insert into ap_expense_report_lines_all).

resize_detail_array:

Make detail array larger if necessary, using realloc.

cost_w_vat

Given a VAT exclusive cost and a vat rate, calculate the cost including vat

(cost with vat = cost + cost*vatrate/100)

get_vatrate

Given a VAT code, get the corresponding VAT rate from vat_code_rates for the latest active date on or before today.

delete_head

For a given invoice, delete a record from ap_expense_report_headers_all. This is called if there was an error in inserting detail records to remove the corresponding header record.

get_oracle_invoice:

Get the nextval from the ap_expense_report_headers_s sequence to use as an Oracle invoice ID (will go into the report_header_id field).

make_dummy_detail:

Insert a record into the ap_expense_report_lines_all table for the case in which there is no detail information. Use the miscellaneous CCID and put the total cost from the invoice header into the amount field.

get_vendor_info:

Select the vendor_id from po_vendors, where vendor_name=(Retek supplier or partner)

Select accts_pay_code_combination_id from po_vendor_sites_all, where vendor_id is the vendor ID just fetched.

Final

Close restart recovery.

I/O Specification

Here is what should go into the Oracle Financials tables. Only the columns that we insert data into are listed—there are many more columns on these tables.

Note: The attribute fields must be set up in Oracle Financials with a code of 'Retek Import' in order for this to work, and a 'RETEK' source must be set up (using Quickcodes).

AP_EXPENSE_REPORT_HEADERS_ALL

COLUMN NAME	NULL	RETEK SOURCE	RETEK TYPE	ORACLE TYPE	COMMENTS
REPORT_HEAD R_ID	NOT NULL	(from ap_expense_report_head ers_s sequence)		NUMBER	Required, primary key
WEEK_END_DA TE		iif_head.inv_c_date	DATE	DATE	
INVOICE_NUM	NOT NULL	iif_head.ext_ref_no (if iif_head.ext_ref_no not exist use iif_head.inv_c_id)	NUMBER(10)	VARCHAR2(50)	
VOUCHNO		Default to '0'		NUMBER	
ATTRIBUTE_CA TEGORY		'Retek Import'		VARCHAR2(15 0)	
ATTRIBUTE1		lif_head.ext_ref_no (rtv_order_no if ext_ref not given; -999 if neither exist)	VARCHAR2(30)	VARCHAR2(15 0)	optional
ATTRIBUTE2		lif_head.payment_ref_n o	VARCHAR2(30)	VARCHAR2(15 0)	optional
ATTRIBUTE3		lif_head.proof_of _delivery_no	VARCHAR2(30)	VARCHAR2(15 0)	optional
ACCOUNTING_D ATE		lif_head.inv_c_date	DATE	DATE	Optional
VENDOR_ID		(from supplier or partner)		NUMBER(15)	Optional
VENDOR_SITE_ ID		addr.oracle_vendor_site _id		NUMBER(15)	Internal ID Foreign key: PO_VENDOR_SIT ES. VENDOR_SITE_I D
TOTAL		sum of cost*qty on iif_detail + sum of non merch amount on iif_non_merch+ sum of cost on iif_merch_vat.. VAT INCLUSIVE.		NUMBER	Required
ACCTS_PAY_CO DE_COMBINATI ON_ID		(from supplier)		NUMBER(15)	

COLUMN NAME	NULL	RETEK SOURCE	RETEK TYPE	ORACLE TYPE	COMMENTS
SET_OF_BOOKS_ID		fif_ora_gl_setup.set_of_books_id	NUMBER(15)	NUMBER(15)	
DEFAULT_CURRENCY_CODE		lif_head.currency_code	VARCHAR2(3)	VARCHAR2(15)	Optional Foreign key: FND_CURRENCIES. CURRENCY_CODE
DEFAULT_EXCHANGE_RATE		1/lif_head.exchange_rate	NUMBER(20, 10)	NUMBER	Foreign key: GL_DAILY_CONVERSION_TYPES. CONVERSION_TYPE
DEFAULT_EXCHANGE_RATE_TYPE		'User' (if an exchange rate is given, else NULL)		VARCHAR2(30)	
DEFAULT_EXCHANGE_DATE		iif_head.invc_date if exchange rate is given, else NULL	DATE	DATE	
DESCRIPTION		iif_head.comments (1st 240 char only)	VARCHAR2(255)	VARCHAR2(240)	
LAST_UPDATE_DATE		iif_head.invc_date	DATE	DATE	
LAST_UPDATED_BY		fif_ora_gl_setup.last_update_ID		NUMBER(15)	Optional
CREATION_DATE		lif_head.invc_date	DATE	DATE	Optional
CREATED_BY		fif_ora_gl_setup.last_update_ID	NUMBER(15)	NUMBER(15)	Optional
SOURCE		'RETEK'		VARCHAR2(80)	Required

AP_EXPENSE_REPORT_LINES_ALL

COLUMN NAME	NULL	RETEK SOURCE	RETEK TYPE	ORACLE TYPE	COMMENTS
REPORT_HEAD ER_ID	NOT NULL	(from ap_expense_report_headers_s sequence)		NUMBER	Required foreign key:
LINE_TYPE_ LOOKUP_ CODE	NOT NULL	'Item', if it is merchandise, (from iif_detail). Can be 'FREIGHT', 'TAX', 'miscellaneous' if it is non- merchandise (from iif_non_merch or iif_merch_vat)		VARCHAR2(25)	Required
SET_OF_BOOKS _ID	NOT NULL	fif_ora_gl_setup.set_of_book s_id	NUMBER(15)	NUMBER(15)	
AMOUNT		lif_detail. payment_unit_cost*payment_ qty, or lif_non_merch.non_merch_ amt or iif_merch_vat total cost .This amount should be VAT inclusive.	NUMBER(20,4)	NUMBER	Required
ITEM_ DESCRIPTION		If merchandise, use SKU description, desc_look. desc_up	VARCHAR2(62)	VARCHAR2(240)	Optional
CODE_COMBIN ATION_ ID		fif_ora_cross_ref.dr_ccid or cr_ccid based on dept class, subclass, location, tran_code and line_type, cost_retail_flag.(where tran_code = 21 will be for invoice)	NUMBER(15)	NUMBER	Internal ID Foreign key: GL_CODE_ COMBINATIO NS.CODE_ COMBINATIO N_ ID
LAST_UPDATE_ DATE		lif_head. invc_date		DATE	Optional
LAST_UPDATE_ BY		fif_ora_gl_setup.last_update_ ID		NUMBER(15)	Optional
CREATED_BY		fif_ora_gl_setup.last_update_ ID		NUMBER(15)	Optional
CREATION_DA TE		lif_head.invc_date		DATE	Optional
ATTRIBUTE_CA TEGORY		'Retek Import'		VARCHAR2(150)	
ATTRIBUTE1		Order number	NUMBER(8)	VARCHAR2(150)	Optional

COLUMN NAME	NULL	RETEK SOURCE	RETEK TYPE	ORACLE TYPE	COMMENTS
ATTRIBUTE2		Shipment number	NUMBER(10)	VARCHAR2(150)	Optional
ATTRIBUTE3		SKU	NUMBER(8)	VARCHAR2(150)	Optional
CURRENCY_CODE		iif_head.currency_code	VARCHAR2(3)	VARCHAR2(15)	

An example of prorates Non_Merch line type: (given by Donald Rome)

Department	Class	Subclass	Item	Non-Merchandise	G/L Account		
1000	100	10	1		1		
1000	100	10	2		1		
1000	100	10		Freight	5101		
1000	100	10		Handling	6101		
1000	100	20	3		2		
1000	100	20		Freight	5102		
1000	100	10		Handling	6102		
1100	100	10	300		3		
1100	100	10		Freight	5103		
1100	100	10		Handling	6103		
Invoice	D145	Supplier	45678				
Description		Quantity	Cost	Extended			
Item	1	125	16	2000			
Item	2	200	15	3000			
Item	3	100	20	2000			
Item	300	300	10	3000			
Merchandise Subtotal				10000			
Freight	Y			600			
Handling	Z			400			
Invoice Total				11000			
Accounting	D-C-S	Merchandise	Merchandise Subtotal	Percentage	Non-Merchandise	Account	Proration

Department	Class	Subclass	Item	Non-Merchandise	G/L Account		
Merchandise	1000-100-10	2000				1	2000
Merchandise	1000-100-10	3000				1	3000
Merchandise	1000-100-20	2000				2	2000
Merchandise	1100-100-10	3000				3	3000
Freight	1000-100-10	5000	10000	0.5	600	5101	300
Freight	1000-100-20	2000	10000	0.2	600	5102	120
Freight	1100-100-10	3000	10000	0.3	600	5103	180
Handling	1000-100-10	5000	10000	0.5	400	6101	200
Handling	1000-100-20	2000	10000	0.2	400	6102	80
Handling	1100-100-10	3000	10000	0.3	400	6103	120
							11000

Generic upload program [fifpdlp107.pc]

Design Overview

This batch program will call the package fifname and pass two arguments to fifname. The argument one will be used to define which package to call bases on the information flow. If information is flowing from financial to retek put 'in' as first argument. The packages will be ora2rms or ps2rms depending on which financial to interfaced with. If information is flowing from retek to financial, put 'out' as first argument. Ether package rms2ora or rms2ps will be called. The second argument will be the function name within the packages.

This programs only process one function each times it runs.

Scheduling Constraints

Processing Cycle: Nightly.

Scheduling Diagram: This process should be scheduled each loading process for that function area in the nightly schedule.

Restart Recovery

Fifpdlp will not be restartble. Should the package return an error code, the batch program should terminate. The package will take care of writing errors to an error table, not in the scope of this document.

Program Flow

Init()

Obtain financial application flag from system_options table. Call restart_init() to set record in the rrestart_bookmark table.

Process()

Combine package name and function name from inline argument passed in base on the financial application flag.

If financial_app = 'P'

 If argument one is 'in' , package name is ps2rms.

 Else package name is rms2ps.

If financial_ap 'O'

 If argument one is 'in', package name is ora2rms.

Else package name is rms2ora.

Call package fifname.

Final()

Call restart_close().

Shared Modules

Listing of all externally referenced functions and Stored procedures and description of usage program_management. Program_to_run(L_program__name, L_error_message).

Function Level Description

I/O Specification

None.

Technical Issues

None.

Terms Interface [fiftrmu2107.pc]

Design Overview

Once `fif_terms` and `fif_freight_terms` are loaded by the Platform Dependent Load Process (PDLP), a validation program (`fiftrmu2107.pc`) will respectively insert/update to the Retek base tables `terms` and `freight_terms`. A row will be written to the appropriate Retek table and subsequently deleted from the staging table only if the necessary data is validated. If the row is invalid, a record will be written to the interface errors table (`if_errors`) and the row will not be deleted from the staging table. Viewing and maintenance of the `if_errors` table and correction of data on `fif_terms`/`fif_freight_terms` are not within the scope of this document.

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Nightly. This process should run prior to any vendor upload.

Pre-Processing: Platform dependent loading process(es) for `fif_terms` and `fif_freight_terms`.

Restart Recovery

Driving Cursor

The logical unit of work for the program will be a single row from the staging tables.

```
select  terms,
        des,
        duedays,
        discdays,
        percent,
        rowid
from    fif_terms
```

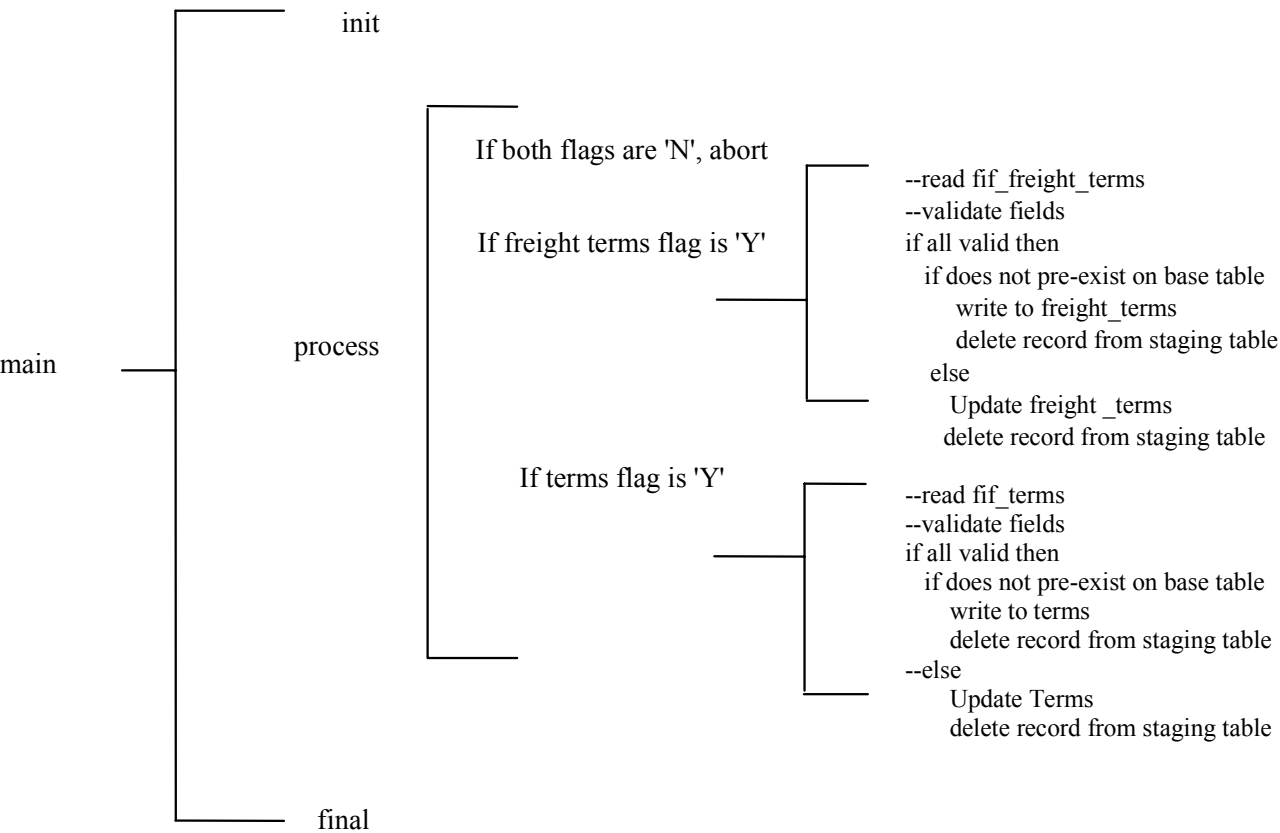
```
select  freight_terms,
        term_desc
        rowid
from    fif_freight_terms;
```

Program Flow

Structure Chart

Validation Upload

This program will select all fields from fif_terms and will validate each field. If a record passes all validation checks, a record will be written/updated to terms. If the validation fails for a specific record, a record will be written to an error table, if_errors, using a separate package documented elsewhere. Records which were written successfully to the terms table will be deleted from fif_terms, while records for which an if_errors record is generated will remain on the staging table. Due to lack of keys on the staging table, the delete should be driven by the fetched rowid value.



Validation:

fif_terms

- terms NOT NULL
- des is NOT NULL

General:

- The rowid selected during the driving cursor should be used in a where clause along with 'terms' to check for duplicate 'keys' on our staging table. This will guard against attempting to insert duplicate keys on Retek's base table.

fif_freight_terms

- freight_terms NOT NULL
- term_desc is NOT NULL

General:

- The rowid selected during the driving cursor should be used in a where clause along with 'freight_terms' to check for duplicate 'keys' on our staging table. This will guard against attempting to insert duplicate keys on Retek's base table.

Defaults:

fif_terms

- duedays -- 0 (Zero)
- discdays -- 0 (Zero)
- percent -- 0 (Zero)

Shared Modules

Interface_sql.insert_interface_errors procedure (ifsqlb.pls) will be used to insert to the interface errors table (if_errors).

Function Level Description

Individual rows not validated and inserted/updated to a base are not deleted from the staging tables and a row is written to the if_errors table.

I/O Specification

None.

Technical Issues

None.

Vendor Interface [fifvndu2107.pc]

Design Overview

Once `fif_sups` and `fif_sups_add` are loaded by the Platform Dependent Load Process (PDLP), a validation program (`fifvndu2.pc`) will update or insert to the Retek tables `sups` and `sups_add`. A row will be written or updated on the Retek tables and subsequently deleted from the staging table only if the data is validated. If the row is invalid, a record will be written to the validation errors table (`if_errors`) and the row will not be deleted from the staging table. Viewing and maintenance of the `if_errors` table and correction of data on `fif_sups/fif_sups_add` are not in the scope of this document.

Because the vendor upload process in general needs to run nightly as well as from the Retek dialog, the main processing will be located in a PL/SQL package. This will allow it to be run in both manners.

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Nightly and on an as-needed basis (started from the Retek dialogue). `Currencies.pc` must run prior to this process (in the event that the new vendor uses a new currency), as well as terms upload, `fiftrmu2107.pc` (in the event that the new vendor uses a new terms code.)

Scheduling Diagram: This process loads new vendors into Retek. It should be scheduled after `fiftrmu2107.pc` in the nightly schedule.

Restart Recovery

Driving Cursor

`Fifvndu2` will not be restartable. Should the package return an error code, the batch program should terminate. The package will take care of writing errors to an error table, not in the scope of this document.

Program Flow

Structure Chart

- Init()
- Process()
 - call package to perform processing
 - if no error then
 - 27) commit
 - else
 - 28) terminate
- final()

Shared Modules

vendval.pls -- package that performs validation of vendors; the main "call" of this batch program

Function Level Description

Will be handled by the package.

I/O Specification

None.

Technical Issues

Should a commit be done even if the validation package returns a failure code?
If not, possible error table inserts will be lost. If so, could we commit bogus data?

Fashion Issues Data Download [fisdnlfd]

Design Overview

fisdnlfd.pc is designed to extract issues data (current and historical) for all forecastable fashion SKUs to an output file. Forecastable SKUs are those with a forecast indicator equal to 'Y' on rag_style for their style. These modules will only send records that meet the following criteria:

- New SKU record with issues (current week only)
- Existing SKU record that has been set to forecastable during a prior week (prior week up to current week only)
- Existing SKU record set to forecastable during week (current and all historical records)
- Existing but reclassified SKU record (current and all historical records)

This module determines which issues records to send from the rag_skus_wh_hist table by referencing the last issues export date field on the rag_skus_wh table. This field indicates the last end of week date that the issues record was sent to Retek Demand Forecasting. As each logical unit of work is processed (SKU/wh combination), the last end of week date unit (from system_variables) is updated into the last export date field.

Note that a SKU WH record will only be written under the following circumstances:

- If the last sales export date is null (new SKU/wh record, existing SKU/wh record set to forecastable, or a reclassification).
- The last sales export date is less equal than the current week being processed.
- Or the load sales indicator on the domain_dept, domain_class, or domain_subclass table has been set to 'Y' (indicating that the system administrator wishes to download a dept, class or subclass).

Once the extract files are written, they will be picked up by Retek Forecasting and loaded into their tables for processing.

Before processing any SKUs, the program must retrieve the domain aggregation level (department, class or subclass). Retek Forecasting has a database size which may necessitate the set up of multiple Retek Forecasting domains (database instances). The client DBA will need to determine at what level (department, class or subclass) this limit is not exceeded for SKU/wh combinations. This level is maintained on system_options so that the correct cursor (dept, class, subclass) can be used during processing. Once this level is determined, the domain_matrix table maintains relationships between the chosen level (department, class or subclass) and the domain.

Scheduling Constraints

Processing Cycle: Weekly

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: By domain ID

Threading set-up: Since threads are determined by the value of the domain ID, the restart_program_status table should contain a row for each domain ID. The thread value of the domain ID should be used as the thread value on this table. The total number of domains/number of threads should be equal to the number of rows on the restart_program_status table. This value must be entered into the restart_control table num_threads field. Note that anytime a new domain is created that an additional row should be added to the restart_program_status table with the thread value equal to the domain ID and the restart_control table num_threads field must be incremented to equal the total number of domains.

Restart Recovery

The logical unit of work is sku and wh for staple records. The commit (update rag_skus_wh, move records from temporary output file to actual output file, truncate temporary file) should occur every 10,000 records.

Program Flow

N/A

Shared Modules

None.

Function Level Description

Init()

- Initialize restart variables.
- Collect system date and domain level information
- Setup output file.

Dept_cursor()

- If the domain level is 'D'ept this function will open, fetch, and close the dept level driving cursor. It will return a 1 when the cursor encounters a NO_DATA_FOUND.

Class_cursor()

- If the domain level is 'C'lass this function will open, fetch, and close the class level driving cursor. It will return a 1 when the cursor encounters a NO_DATA_FOUND.

Subclass_cursor()

- If the domain level is 'S'ubclass this function will open, fetch, and close the subclass level driving cursor. It will return a 1 when the cursor encounters a NO_DATA_FOUND.

Process()

- The function declares structures to hold data fetched from the driving cursor and for the array update of RAG_SKUS_WH.LAST_ISSUES_EXPORT_DATE.
- It calls the size_driv_cur_struct and size_update_sturct function to allocate memory for these structures.
- Depending on the domain level it will call dept_cursor, class_cursor or subclass_cursor to open the needed driving cursor.
- It will then enter a while loop that terminates when one of the driving cursor functions returns a 1. This loop will contain a call to dept_cursor, class_cursor or subclass_cursor to fetch the needed driving cursor and a call to process_records to handle the records returned by the driving cursor.
- After the driving cursor loop terminates, it calls the post_wh function to update any rag_skus_wh records that didn't get processed by process_records.

Process_records()

- The function loops through the driving cursor array and processes records it contains.
 - If either the sku or the wh changes, the wh update array is filled with the previous sku/wh rowid's on rag_skus_wh. If the addition of this record fills the update array it will be posted to the database by calling the post_wh function.
 - If the current counter is greater than the max counter, call to restart_commit will be made. A call to post_wh is done before this to make sure any records in the output file are updated on rag_skus_wh before a commit is performed.
 - Copy the current rowid, sku, and wh into the previous holder variables.
 - Call write_fashion_file to write the current records information to the output file.
- If the driving cursor returned a one, copy the previous rowid into the update array.

Post_wh()

- This function will use the update array populated by process_records to update rag_skus_wh.last_issues_export_date.

Write_fashion_file()

- This function writes the current record's information to the temporary output file.
- Size_driv_cur_struct()
- Allocates memory for the driving cursor fetch array. It sizes the array to g_l_restart_max_counter.

Size_update_struct()

- Allocates memory for the win_wh update array. It sizes the array to g_l_restart_max_counter.

Final()

- Close restart recovery process
- Clean up output files.

I/O Specification

Fissuenn.dat – issue output file, where nn is the domain (i.e. thread).

Example record:

1234567890123456789012345678901234567890123456781234567890123

3 10000004 20000102 108.01

Field Name	Field Type	Default Value	Description
WH	Char(20)		WH identifier – left-justified, right-padded with spaces.
SKU	Char(20)		RMS item identifier – left-justified, right-padded with spaces.
End of week date	Char(8)		End of week date for record
Issues Quantity	Number(13)		Quantity of issues at wh for the sku for the end of week date. Decimal is included in output file – right-justified, left-padded with spaces.

Staple Issues Data Download [fisdnllds]

Design Overview

fisdnllds.pc is designed to extract issues data (current and historical) for all forecastable staple SKUs to an output file. Forecastable SKUs are those with a forecast indicator equal to 'Y' on win_skus. These modules will only send records that meet the following criteria:

- New SKU record with issues (current week only).
- Existing SKU record that has been set to forecastable during a prior week (prior week up to current week only).
- Existing SKU record set to forecastable during week (current and all historical records).
- Existing but reclassified SKU record (current and all historical records).

This module determines which issues records to send from the win_wh_hist table by referencing the last issues export date field on the win_wh table. This field indicates the last end of week date that the sales record was sent to Retek Demand Forecasting. As each logical unit of work is processed (SKU/wh combination), the last end of week date unit (from system_variables) is updated into the last export date field.

Note that a SKU/wh record will only be written under the following circumstances:

- If the last issues export date is null (new SKU/wh record, existing SKU/wh record set to forecastable, or a reclassification).
- The last sales export date is less than or equal to the current week being processed.
- Or the load sales indicator on the domain_dept, domain_class, or domain_subclass table has been set to 'Y' (indicating that the system administrator wishes to download a dept, class or subclass).

Once the extract files are written, they will be picked up by Retek Forecasting and loaded into their tables for processing.

Before processing any SKUs, the program must retrieve the domain aggregation level (department, class or subclass). Retek Forecasting has a database size which may necessitate the set up of multiple Retek Forecasting domains (database instances). The client DBA will need to determine at what level (department, class or subclass) this limit is not exceeded for SKU/wh combinations. This level is maintained on system_options so that the correct cursor (dept, class, subclass) can be used during processing. Once this level is determined, the domain_matrix table maintains relationships between the chosen level (department, class or subclass) and the domain.

Scheduling Constraints

Processing Cycle:	Weekly
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	By domain ID

Threading set-up: Since threads are determined by the value of the domain ID, the restart_program_status table should contain a row for each domain ID. The thread value of the domain ID should be used as the thread value on this table. The total number of domains/number of threads should be equal to the number of rows on the restart_program_status table. This value must be entered into the restart_control table num_threads field. Note that anytime a new domain is created that an additional row should be added to the restart_program_status table with the thread value equal to the domain ID and the restart_control table num_threads field must be incremented to equal the total number of domains.

Restart Recovery

The logical unit of work is sku and wh for staple records. The commit (update win_wh, move records from temporary output file to actual output file, truncate temporary file) should occur every 10,000 records.

Program Flow

N/A

Shared Modules

None.

Function Level Description

Init()

- Initialize restart variables.
- Collect system date and domain level information
- Setup output file.

Dept_cursor()

- If the domain level is 'D'ept this function will open, fetch, and close the dept level driving cursor. It will return a 1 when the cursor encounters a NO_DATA_FOUND.

Class_cursor()

- If the domain level is 'C'lass this function will open, fetch, and close the class level driving cursor. It will return a 1 when the cursor encounters a NO_DATA_FOUND.

Subclass_cursor()

- If the domain level is 'S'ubclass this function will open, fetch, and close the subclass level driving cursor. It will return a 1 when the cursor encounters a NO_DATA_FOUND.

Process()

- The function declares structures to hold data fetched from the driving cursor and for the array update of WIN_WH.LAST_ISSUES_EXPORT_DATE.
- It calls the size_driv_cur_struct and size_update_sturct function to allocate memory for these structures.
- Depending on the domain level it will call dept_cursor, class_cursor or subclass_cursor to open the needed driving cursor.
- It will then enter a while loop that terminates when one of the driving cursor functions returns a 1. This loop will contain a call to dept_cursor, class_cursor or subclass_cursor to fetch the needed driving cursor and a call to process_records to handle the records returned by the driving cursor.
- After the driving cursor loop terminates, it calls the post_wh function to update any win_wh records that didn't get processed by process_records.

Process_records()

- The function loops through the driving cursor array and processes records it contains.
 - If either the sku or the wh changes, the wh update array is filled with the previous sku/wh rowid's on win_wh. If the addition of this record fills the update array it will be posted to the database by calling the post_wh function.
 - If the current counter is greater than the max counter, call to restart_commit will be made. A call to post_wh is done before this to make sure any records in the output file are updated on win_wh before a commit is performed.
 - Copy the current rowid, sku, and wh into the previous holder variables.
 - Call write_staple_file to write the current records information to the output file.
- If the driving cursor returned a one copy the previous rowid into the update array.

Post_wh()

- This function will use the update array populated by process_records to update win_wh.last_issues_export_date

Write_staple_file()

- This function writes the current record's information to the temporary output file.

Size_driv_cur_struct()

- Allocates memory for the driving cursor fetch array. It sizes the array to g_l_restart_max_counter.

Size_update_struct()

- Allocates memory for the win_wh update array. It sizes the array to g_l_restart_max_counter.

Final()

- Close restart recovery process
- Clean up output files.

I/O Specification

Sissuenn.dat – issue output file, where *nn* is the domain (i.e. thread).

Example record:

1234567890123456789012345678901234567890123456781234567890123

3 10000004 20000102 108.01

Field Name	Field Type	Default Value	Description
WH	Char(20)		WH identifier – left-justified, right-padded with spaces.
SKU	Char(20)		RMS item identifier – left-justified, right-padded with spaces.
End of week date	Char(8)		End of week date for record
Issues Quantity	Number(13)		Quantity of issues at wh for the sku for the end of week date. Decimal is included in output file – right-justified, left-padded with spaces.

Fashion Merchandise Hierarchy Download [fmednldf]

Design Overview

fmednldf.pc is designed to extract the product hierarchy for all in the SKUs into output file. These modules will send the full merchandise hierarchy to each domain each night since changes could occur on a daily basis (clients may schedule these batch programs, at their discretion, to run daily, weekly, etc.).

Before processing styles, the program must retrieve the domain aggregation level (department, class or subclass). Retek Forecasting has a size limitation for the multidimensional database, so they may declare multiple instances (or domains) of the database to optimize performance. The client DBA will need to determine at what level (department, class or subclass) this limit is not exceeded for SKU/store combinations. This level is maintained on system_options so that the correct cursor (dept, class, subclass) can be used during processing. Once this level is determined, the domain_dept, domain_class, and domain_subclass tables maintain relationships between the chosen level (department, class or subclass) and the domain.

Scheduling Constraints

Processing Cycle:	Daily
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	DEPT

Restart Recovery

The logical unit of work is SKU. The commit (move records from temporary output file to actual output file, truncate temporary file) should occur every 10,000 records.

Fashion SKU Driving Cursors:

declare cursors - all ticks (') and quotes (") are replaced with spaces, fashion sku description is cut back to 50 characters

- Department

EXEC SQL DECLARE C_fashion_dept CURSOR FOR

```
SELECT lpad(to_char(rag_skus.sku),8,'0'),
       lpad(to_char(rag_skus.style),8,'0'),
       REPLACE(REPLACE(rag_style.style_desc,'"',' '),"'',' '),
       lpad(to_char(colour.colour),4,'0'),
       REPLACE(REPLACE(colour.colour_desc,'"',' '),"'',' '),
       rag_skus.size1,
       REPLACE(REPLACE(s1.size_desc,'"',' '),"'',' '),
       NVL(rag_skus.size2,' '),
       REPLACE(REPLACE(NVL(s2.size_desc,' '),"'',' '),"'',' '),
       lpad(to_char(rag_style.dept),4,'0')||
       lpad(to_char(rag_style.class),4,'0')||
       lpad(to_char(rag_style.subclass),4,'0'),
       REPLACE(REPLACE(subclass.sub_name,'"',' '),"'',' '),
       lpad(to_char(rag_style.dept),4,'0')||
       lpad(to_char(rag_style.class),4,'0'),
       REPLACE(REPLACE(class.class_name,'"',' '),"'',' '),
       lpad(to_char(rag_style.dept),4,'0'),
       REPLACE(REPLACE(deps.dept_name,'"',' '),"'',' '),
```

```
lpad(to_char(deps.group_no),4,'0'),
REPLACE(REPLACE(groups.group_name,',',' '),',',' '),
lpad(to_char(groups.division),4,'0'),
REPLACE(REPLACE(division.div_name,',',' '),',',' '),
lpad(to_char(its.supplier),10,'0'),
REPLACE(REPLACE(sups.sup_name,',',' '),',',' '),
rag_style.forecast_ind,
lpad(to_char(NVL(domain_dept.domain_id, 0)),2,'0')
FROM rag_skus,
rag_style,
subclass,
class,
deps,
colour,
groups,
division,
domain_dept,
item_supplier its,
sups,
sizes s1,
sizes s2,
v_restart_dept
WHERE v_restart_dept.driver_name = 'DEPT'
AND v_restart_dept.num_threads = :pi_num_threads
AND v_restart_dept.thread_val = :pi_thread_val
AND v_restart_dept.driver_value = rag_style.dept
```

```

AND rag_style.style = rag_skus.style
AND rag_skus.size1 = s1.size_id
AND rag_skus.size2 = s2.size_id(+)
AND rag_skus.colour = colour.colour
AND rag_style.dept = deps.dept
AND rag_style.dept = domain_dept.dept(+)
AND rag_style.dept = class.dept
AND rag_style.class = class.class
AND rag_style.dept = subclass.dept
AND rag_style.class = subclass.class
AND rag_style.subclass = subclass.subclass
AND deps.group_no = groups.group_no
AND groups.division = division.division
AND rag_skus.sku > NVL(:os_restart_sku, -999)
AND its.item = rag_skus.sku
AND its.primary_supp_ind = 'Y'
AND its.supplier = sups.supplier

ORDER BY rag_skus.sku;

```

- Class

```

EXEC SQL DECLARE C_fashion_class CURSOR FOR
SELECT lpad(to_char(rag_skus.sku),8,'0'),
       lpad(to_char(rag_skus.style),8,'0'),
       REPLACE(REPLACE(rag_style.style_desc,' ',''),',''),
       lpad(to_char(colour.colour),4,'0'),
       REPLACE(REPLACE(colour.colour_desc,' ',''),',''),
       rag_skus.size1,

```

```
REPLACE(REPLACE(s1.size_desc,' ',''),',''),
NVL(rag_skus.size2,''),
REPLACE(REPLACE(NVL(s2.size_desc,' '),',''),',''),
lpad(to_char(rag_style.dept),4,'0')||
lpad(to_char(rag_style.class),4,'0')||
lpad(to_char(rag_style.subclass),4,'0'),
REPLACE(REPLACE(subclass.sub_name,' ',''),',''),
lpad(to_char(rag_style.dept),4,'0')||
lpad(to_char(rag_style.class),4,'0'),
REPLACE(REPLACE(class.class_name,' ',''),',''),
lpad(to_char(rag_style.dept),4,'0'),
REPLACE(REPLACE(deps.dept_name,' ',''),',''),
lpad(to_char(deps.group_no),4,'0'),
REPLACE(REPLACE(groups.group_name,' ',''),',''),
lpad(to_char(groups.division),4,'0'),
REPLACE(REPLACE(division.div_name,' ',''),',''),
lpad(to_char(its.supplier),10,'0'),
REPLACE(REPLACE(sups.sup_name,' ',''),',''),
rag_style.forecast_ind,
lpad(to_char(NVL(domain_class.domain_id, 0)),2,'0')
FROM rag_skus,
rag_style,
subclass,
class,
deps,
colour,
```



```
groups,
division,
domain_class,
item_supplier its,
supps,
sizes s1,
sizes s2,
v_restart_dept
WHERE v_restart_dept.driver_name = 'DEPT'
AND v_restart_dept.num_threads = :pi_num_threads
AND v_restart_dept.thread_val = :pi_thread_val
AND v_restart_dept.driver_value = rag_style.dept
AND rag_style.style = rag_skus.style
AND rag_skus.size1 = s1.size_id
AND rag_skus.size2 = s2.size_id(+)
AND rag_skus.colour = colour.colour
AND rag_style.dept = deps.dept
AND rag_style.dept = domain_class.dept(+)
AND rag_style.class = domain_class.class(+)
AND rag_style.dept = class.dept
AND rag_style.class = class.class
AND rag_style.dept = subclass.dept
AND rag_style.class = subclass.class
AND rag_style.subclass = subclass.subclass
AND deps.group_no = groups.group_no
AND groups.division = division.division
```

```
AND rag_skus.sku > NVL(:os_restart_sku, -999)
```

```
AND its.item = rag_skus.sku
```

```
AND its.primary_supp_ind = 'Y'
```

```
AND its.supplier = sups.supplier
```

```
ORDER BY rag_skus.sku;
```

- Subclass

```
EXEC SQL DECLARE C_fashion_subclass CURSOR FOR
```

```
SELECT lpad(to_char(rag_skus.sku),8,'0'),  
       lpad(to_char(rag_skus.style),8,'0'),  
       REPLACE(REPLACE(rag_style.style_desc,' ',''),''',''),  
       lpad(to_char(colour.colour),4,'0'),  
       REPLACE(REPLACE(colour.colour_desc,' ',''),''',''),  
       rag_skus.size1,  
       REPLACE(REPLACE(s1.size_desc,' ',''),''',''),  
       NVL(rag_skus.size2,' '),  
       REPLACE(REPLACE(NVL(s2.size_desc,' '),''',''),''',''),  
       lpad(to_char(rag_style.dept),4,'0')||  
       lpad(to_char(rag_style.class),4,'0')||  
       lpad(to_char(rag_style.subclass),4,'0'),  
       REPLACE(REPLACE(subclass.sub_name,' ',''),''',''),  
       lpad(to_char(rag_style.dept),4,'0')||  
       lpad(to_char(rag_style.class),4,'0'),  
       REPLACE(REPLACE(class.class_name,' ',''),''',''),  
       lpad(to_char(rag_style.dept),4,'0'),  
       REPLACE(REPLACE(deps.dept_name,' ',''),''',''),  
       lpad(to_char(deps.group_no),4,'0'),
```

```

        REPLACE(REPLACE(groups.group_name,' ',''),',' ',''),
        lpad(to_char(groups.division),4,'0'),
        REPLACE(REPLACE(division.div_name,' ',''),',' ',''),
        lpad(to_char(its.supplier),10,'0'),
        REPLACE(REPLACE(sups.sup_name,' ',''),',' ',''),
        rag_style.forecast_ind,
        lpad(to_char(NVL(domain_subclass.domain_id, 0)),2,'0')
FROM rag_skus,
     rag_style,
     subclass,
     class,
     deps,
     colour,
     groups,
     division,
     domain_subclass,
     item_supplier its,
     sups,
     sizes s1,
     sizes s2,
     v_restart_dept
WHERE v_restart_dept.driver_name = 'DEPT'
      AND v_restart_dept.num_threads = :pi_num_threads
      AND v_restart_dept.thread_val = :pi_thread_val
      AND v_restart_dept.driver_value = rag_style.dept
      AND rag_style.style = rag_skus.style

```

```
AND rag_skus.size1 = s1.size_id
AND rag_skus.size2 = s2.size_id(+)
AND rag_skus.colour = colour.colour
AND rag_style.dept = deps.dept
AND rag_style.dept = domain_subclass.dept(+)
AND rag_style.class = domain_subclass.class(+)
AND rag_style.subclass = domain_subclass.subclass(+)
AND rag_style.dept = class.dept
AND rag_style.class = class.class
AND rag_style.dept = subclass.dept
AND rag_style.class = subclass.class
AND rag_style.subclass = subclass.subclass
AND deps.group_no = groups.group_no
AND groups.division = division.division
AND rag_skus.sku > NVL(:os_restart_sku, -999)
AND its.item = rag_skus.sku
AND its.primary_supp_ind = 'Y'
AND its.supplier = sups.supplier

ORDER BY rag_skus.sku;
```

Program Flow

Pseudo-code (note functions shown in-line, should be defined outside main)

```
main(int argc, char* argv[])
```

```
{
```

```
    extract login ID, password, thread value from input arguments
```

```
    declare_fashion_cursor(pass_thread_value);
```

```
open_fashion_cursor(pass_thread_value);
fetch_fashion_cursor(pass_thread_value);

while(1)
{
    if record counter < global maximum counter
        write to product master (merch hier.) temp file
        increment counter of records written
    else
        call restart logic to cat temp Prod. Master file to
final file
        reset counter of records written
    end-if

    fetch_fashion_cursor(pass_thread_value);
}
```

Function declarations

```
declare_fashion_cursor(pass_thread_value)
{
    switch pass_thread_value
    {
        case 'D'
            declare f_domain_dept cursor
        case 'C'
```

```
        declare f_domain_class cursor
    case 'S'
        declare f_domain_subclass cursor
    default:
        sprintf(err_date, "undefined level %c for declare
fashion cursor", pass_thread_value)
    }
    if SQL error found
        sprintf(err_date, "declare %c_level fashion cursor",
pass_thread_value)
    }

    open_fashion_cursor(pass_thread_value)
    {
        switch pass_thread_value
        {
            case 'D':
                open f_domain_dept cursor
            case 'C':
                open f_domain_class cursor
            case 'S':
                open f_domain_subclass cursor
            default:
                sprintf(err_date, "undefined level %c for open
fashion cursor", pass_thread_value)
        }
    }
```

```
        if SQL error found

            sprintf(err_date, "open %c_level fashion cursor",
pass_thread_value)
        }

fetch_fashion_cursor(pass_thread_value)
{
    switch pass_thread_value
    {
        case 'D':

            open f_domain_dept cursor
        case 'C':

            open f_domain_class cursor
        case 'S':

            open f_domain_subclass cursor
        default:

            sprintf(err_date, "undefined level %c for open
fashion cursor", pass_thread_value)
    }

    if SQL error found

        sprintf(err_date, "fetcg %c_level fashion cursor",
pass_thread_value)
    }

close_fashion_cursor(pass_thread_value)
{
```

```
switch pass_thread_value
{
case 'D':
    close f_domain_dept cursor
case 'C':
    close f_domain_class cursor
case 'S':
    close f_domain_subclass cursor
default:
    sprintf(err_date, "undefined level %c for close
fashion cursor", pass_thread_value)
}

if SQL error found
    sprintf(err_date, "close %c_level fashion cursor",
pass_thread_value)
}
```

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

Output files fmhiernn.date (where *nn* equals the thread value – this is not equal to the domain_id)

SKU	Varchar2	20
SKU desc	Varchar2	30
Color code	Varchar2	4
Color desc	Varchar2	24
Size 1 code	Varchar2	6
Size 1 desc	Varchar2	30
Size 2 code	Varchar2	6
Size 2 desc	Varchar2	30
Style	Varchar2	20
Style desc	Varchar2	40
Subclass	Varchar2	20 – dept+class+subclass
Subclass desc	Varchar2	40
Class	Varchar2	20 – dept+class
Class desc	Varchar2	40
Dept	Varchar2	20 – dept
Dept desc	Varchar2	40
Group	Varchar2	20
Group name	Varchar2	40
Division	Varchar2	20
Division name	Varchar2	40
Supplier	Varchar2	20
Supplier name	Varchar2	40
Forecast Indicator	Varchar2	1
Domain ID	Varchar2	2

Staple Merchandise Hierarchy Download [fmednlds]

Design Overview

fmednlds.pc is designed to extract the product hierarchy for all staple SKUs into output file. These modules will send the full merchandise hierarchy to each domain each night since changes could occur on a daily basis (clients may schedule these batch programs, at their discretion, to run daily, weekly, etc.).

Before processing any SKUs, the program must retrieve the domain aggregation level (department, class or subclass). Retek Forecasting has a size limitation for the multidimensional database, so they may declare multiple instances (or domains) of the database to optimize performance. The client DBA will need to determine at what level (department, class or subclass) this limit is not exceeded for SKU/store combinations. This level is maintained on system_options so that the correct cursor (dept, class, subclass) can be used during processing. Once this level is determined, the domain_dept, domain_class, and domain_subclass tables maintain relationships between the chosen level (department, class or subclass) and the domain.

Scheduling Constraints

Processing Cycle:	Daily
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	DEPT

Restart Recovery

The logical unit of work is SKU. The commit (move records from temporary output file to actual output file, truncate temporary file) should occur every 10,000 records.

Staple SKU driving cursors:

Declare cursors - all ticks (') and quotes (") are replaced with spaces

- Department

EXEC SQL DECLARE C_staple_dept CURSOR FOR

```
SELECT lpad(to_char(win_skus.sku),8,'0'),
       REPLACE(REPLACE(win_skus.sku_desc,' ',' '),',' '),
       lpad(to_char(win_skus.dept),4,'0')||
       lpad(to_char(win_skus.class),4,'0')||
       lpad(to_char(win_skus.subclass),4,'0'),
       REPLACE(REPLACE(subclass.sub_name,' ',' '),',' '),
       lpad(to_char(win_skus.dept),4,'0')||
       lpad(to_char(win_skus.class),4,'0'),
       REPLACE(REPLACE(class.class_name,' ',' '),',' '),
       lpad(to_char(win_skus.dept),4,'0'),
       REPLACE(REPLACE(deps.dept_name,' ',' '),',' '),
       lpad(to_char(deps.group_no),4,'0'),
       REPLACE(REPLACE(groups.group_name,' ',' '),',' '),
       lpad(to_char(groups.division),4,'0'),
       REPLACE(REPLACE(division.div_name,' ',' '),',' '),
       lpad(to_char(item_supplier.supplier),10,'0'),
       REPLACE(REPLACE(sups.sup_name,' ',' '),',' '),
       win_skus.forecast_ind,
       lpad(to_char(domain_dept.domain_id),2,'0')
```

```
FROM win_skus,
      subclass,
      class,
      deps,
      groups,
      division,
      item_supplier,
      sups,
      domain_dept,
      v_restart_dept
WHERE v_restart_dept.driver_name = 'DEPT'
      AND v_restart_dept.num_threads = :pi_num_threads
      AND v_restart_dept.thread_val = :pi_thread_val
      AND v_restart_dept.driver_value = win_skus.dept
      AND win_skus.dept = deps.dept
      AND win_skus.dept = domain_dept.dept(+)
      AND win_skus.dept = class.dept
      AND win_skus.class = class.class
      AND win_skus.dept = subclass.dept
      AND win_skus.class = subclass.class
      AND win_skus.subclass = subclass.subclass
      AND win_skus.sku = item_supplier.item
      AND item_supplier.supplier = sups.supplier
      AND item_supplier.primary_supp_ind = 'Y'
      AND deps.group_no = groups.group_no
      AND groups.division = division.division
```

```

AND win_skus.sku > NVL(:os_restart_sku, -999)

ORDER BY win_skus.sku;

• Class

EXEC SQL DECLARE C_staple_class CURSOR FOR

SELECT lpad(to_char(win_skus.sku),8,'0'),

        REPLACE(REPLACE(win_skus.sku_desc,',',''),'','),

        lpad(to_char(win_skus.dept),4,'0')||

        lpad(to_char(win_skus.class),4,'0')||

        lpad(to_char(win_skus.subclass),4,'0'),

        REPLACE(REPLACE(subclass.sub_name,',',''),'','),

        lpad(to_char(win_skus.dept),4,'0')||

        lpad(to_char(win_skus.class),4,'0'),

        REPLACE(REPLACE(class.class_name,',',''),'','),

        lpad(to_char(win_skus.dept),4,'0'),

        REPLACE(REPLACE(deps.dept_name,',',''),'','),

        lpad(to_char(deps.group_no),4,'0'),

        REPLACE(REPLACE(groups.group_name,',',''),'','),

        lpad(to_char(groups.division),4,'0'),

        REPLACE(REPLACE(division.div_name,',',''),'','),

        lpad(to_char(item_supplier.supplier),10,'0'),

        REPLACE(REPLACE(sups.sup_name,',',''),'','),

        win_skus.forecast_ind,

        lpad(to_char(domain_class.domain_id),2,'0')

FROM win_skus,

        subclass,

        class,

```

```
        deps,  
        groups,  
        division,  
        item_supplier,  
        sups,  
        domain_class,  
        v_restart_dept  
WHERE v_restart_dept.driver_name = 'DEPT'  
      AND v_restart_dept.num_threads = :pi_num_threads  
      AND v_restart_dept.thread_val = :pi_thread_val  
      AND v_restart_dept.driver_value = win_skus.dept  
      AND win_skus.dept = deps.dept  
      AND win_skus.dept = domain_class.dept(+)  
      AND win_skus.class = domain_class.class(+)  
      AND win_skus.dept = class.dept  
      AND win_skus.class = class.class  
      AND win_skus.dept = subclass.dept  
      AND win_skus.class = subclass.class  
      AND win_skus.subclass = subclass.subclass  
      AND win_skus.sku = item_supplier.item  
      AND item_supplier.supplier = sups.supplier  
      AND item_supplier.primary_supp_ind = 'Y'  
      AND deps.group_no = groups.group_no  
      AND groups.division = division.division  
      AND win_skus.sku > NVL(:os_restart_sku, -999)  
ORDER BY win_skus.sku;
```

- Subclass

```
EXEC SQL DECLARE C_staple_subclass CURSOR FOR

SELECT lpad(to_char(win_skus.sku),8,'0'),

       REPLACE(REPLACE(win_skus.sku_desc,',',' '),',',' '),

       lpad(to_char(win_skus.dept),4,'0')||

       lpad(to_char(win_skus.class),4,'0')||

       lpad(to_char(win_skus.subclass),4,'0'),

       REPLACE(REPLACE(subclass.sub_name,',',' '),',',' '),

       lpad(to_char(win_skus.dept),4,'0')||

       lpad(to_char(win_skus.class),4,'0'),

       REPLACE(REPLACE(class.class_name,',',' '),',',' '),

       lpad(to_char(win_skus.dept),4,'0'),

       REPLACE(REPLACE(deps.dept_name,',',' '),',',' '),

       lpad(to_char(deps.group_no),4,'0'),

       REPLACE(REPLACE(groups.group_name,',',' '),',',' '),

       lpad(to_char(groups.division),4,'0'),

       REPLACE(REPLACE(division.div_name,',',' '),',',' '),

       lpad(to_char(item_supplier.supplier),10,'0'),

       REPLACE(REPLACE(sups.sup_name,',',' '),',',' '),

       win_skus.forecast_ind,

       lpad(to_char(domain_subclass.domain_id),2,'0')

FROM win_skus,

     subclass,

     class,

     deps,

     groups,
```

```
        division,
        item_supplier,
        sups,
        domain_subclass,
        v_restart_dept
WHERE v_restart_dept.driver_name = 'DEPT'

      AND v_restart_dept.num_threads = :pi_num_threads

      AND v_restart_dept.thread_val = :pi_thread_val

      AND v_restart_dept.driver_value = win_skus.dept

      AND win_skus.dept = deps.dept

      AND win_skus.dept = domain_subclass.dept(+)

      AND win_skus.class = domain_subclass.class(+)

      AND win_skus.subclass = domain_subclass.subclass(+)

      AND win_skus.dept = class.dept

      AND win_skus.class = class.class

      AND win_skus.dept = subclass.dept

      AND win_skus.class = subclass.class

      AND win_skus.subclass = subclass.subclass

      AND win_skus.sku = item_supplier.item

      AND item_supplier.supplier = sups.supplier

      AND item_supplier.primary_supp_ind = 'Y'

      AND deps.group_no = groups.group_no

      AND groups.division = division.division

      AND win_skus.sku > NVL(:os_restart_sku,-999)

ORDER BY win_skus.sku;
```


Program Flow

Pseudo-code (note functions shown in-line, should be defined outside main)

```
main(int argc, char* argv[])
{
    extract login ID, password, thread value from input arguments

    declare_staple_cursor(pass_thread_value);
    open_staple_cursor(pass_thread_value);
    fetch_staple_cursor(pass_thread_value);

    while(1)
    {
        if record counter < global maximum counter
            write to product master (merch hier.) temp file
            increment counter of records written
        else
            call restart logic to cat temp Prod. Master file to
            final file
            reset counter of records written
        end-if

        fetch_staple_cursor(pass_thread_value);}
}
```

Function declarations

```
declare_staple_cursor(pass_thread_value) – job department/thread value
{
    switch pass_thread_value
    {
        case 'D':
            declare s_domain_dept cursor
        case 'C':
            declare s_domain_class cursor
        case 'S':
            declare s_domain_subclass cursor
        default:
            sprintf(err_date, “undefined level %c for declare
staple cursor”, pass_thread_value)
    }

    if SQL error found
        sprintf(err_date, “declare %c_level staple cursor”,
pass_thread_value)
}

open_staple_cursor(pass_thread_value)
{
    switch pass_thread_value
    {
        case 'D':
            open s_domain_dept cursor
        case 'C':
```

```
        open s_domain_class cursor
    case 'S':
        open s_domain_subclass cursor
    default:
        sprintf(err_date, "undefined level %c for open
staple cursor", pass_thread_value)
    }
    if SQL error found
        sprintf(err_date, "open %c_level staple cursor",
pass_thread_value)
    }

    fetch_staple_cursor(pass_thread_value)
    {
        switch pass_thread_value
        {
            case 'D':
                open s_domain_dept cursor
            case 'C':
                open s_domain_class cursor
            case 'S':
                open s_domain_subclass cursor
            default:
                sprintf(err_date, "undefined level %c for open
staple cursor", pass_thread_value)
        }
    }
```

```
        if SQL error found
            sprintf(err_date, "fetch %c_level staple cursor",
pass_thread_value)
        }

open_staple_cursor(pass_thread_value)
{
    switch pass_thread_value
    {
        case 'D':
            open s_domain_dept cursor
        case 'C':
            open s_domain_class cursor
        case 'S':
            open s_domain_subclass cursor
        default:
            sprintf(err_date, "undefined level %c for open
staple cursor", pass_thread_value)
    }

    if SQL error found
        sprintf(err_date, "open %c_level staple cursor",
pass_thread_value)
    }
```

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

Output file smhiernn.dat (where *nn* equals the thread value – this is not the domain_id)

SKU	Varchar2	20	Populate with SKU
SKU desc	Varchar2	70	Populate with SKU description
Style	Varchar2	20	NULL
SKU desc	Varchar	240	
Subclass	Varchar2	20	dept+class+subclass
Subclass desc	Varchar2	40	
Class	Varchar2		20 – dept+class
Class desc	Varchar2	40	
Dept	Varchar2	20	dept
Dept desc	Varchar2	40	
Group	Varchar2	20	
Group name	Varchar2	40	
Division	Varchar2	20	
Division name	Varchar2	40	
Supplier	Varchar2	20	
Supplier name	Varchar2	40	
Forecast Indicator	Varchar2		1
Domain ID	Varchar2		2

Organizational Hierarchy Download [forgdnld]

Design Overview

forgdnld.pc is designed to build the organizational hierarchy for each store. The standard Retek hierarchy (district, region, area, chain, company) is written for each store. In addition, store format and store class are passed in this file each as a separate roll up level. Once the extract file is completed, it will be picked up by Retek Forecasting and loaded into their tables for processing. This module will also create a store attribute file containing each store's open and close date.

Retek Forecasting has a database size which may necessitate the set up of multiple Retek Forecasting domains (database instances). Fortunately, the segmentation level is held in the merchandise, not product hierarchy. Therefore, only one hierarchy file will be created and Retek Forecasting will load a copy of this file into all domains.

Note: Since only one organizational or product hierarchy is created, then the standard threading logic algorithm to allocate threads will be used based on the logical unit of work, which is region.

The entire set of stores and store attributes will be sent to Retek Forecasting each day. Retek Forecasting will determine any changes as it loads this module's extract and will update records in each domain as needed.

An organizational hierarchy extract currently exists that is used for interfacing information between RMS and RDF. One of the output files of this extract includes the store number, store open date and store close date for all stores within the system. This extract will be modified to also include the selling space within each store. In this manner, forgdnld.pc can be run when either or both applications (RDF v. RIPP) are in use.

Scheduling Constraints

Processing Cycle:	Daily
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	Region

Restart Recovery

The logical unit of work is region for all records. Since this is an output based program that is not committing information to tables, the commit (move records from temporary output file to actual output file and truncate temporary file) should occur every 50 records.

- Product Hierarchy Cursor

```
Select  st.store, st.store_name, st.store_open_date, st.store_close_date, st.district,
ds.district_name, ds.region, rg.region_name, rg.area, ar.area_name, ar.chain,
ch.chain_name, co.company, co.company_name, st.store_format,
sf.format_name, st.store_class
```

```
From   store st, district ds, region rg, area ar, chain ch, comphead co,
store_format sf
```

```
Where  st.district = ds.district
```

```
and    ds.region = rg.region
```

```
and    rg.area = ar.area
```

```
and    ar.chain = ch.chain
```

```
and    st.store_format = sf.store_format
```

```
and    ((rg.region > NVL(:ora_restart_region, rg.region - 1) or
        (rg.region = :ora_restart_region))
```

```
Order by      rg.region
```

Program Flow

Pseudo-code -- Determine domain level and maximum number of domains for staple & fashion items

Refer to RESTART_FILE skeleton for restart logic and variable definitions

```
strcpy (PROGRAM, "forgdnld.pc");
```

```
strcpy(g_s_restart_name, "restart_file");
```

```
FILE * stmstrfp
```

```
FILE * stattrfp
```

```
char temp_location_master[30]
```

```
char final_location_master[30]
```

```
char temp_location_attributes[30]
```

```
char final_location_attributes[30]
```

open files for output

```
sprintf(temp_location_master, "lmast.tmp");
```

```
sprintf(final_location_master, "lmast.dat");
```

```
sprintf(temp_location_attributes, "lattr.tmp");
```

```
sprintf(final_location_attributes, "lattr.dat");
```

```
if ((outfp = fopen(temp_location_master, "w")) == NULL) return(-1);
```

```
if ((outfp = fopen(temp_location_attributes, "w")) == NULL) return(-1);
```

Declare cursor

Open cursor

Fetch first row into local variables:

```
store
```

```
store_desc
```

```
store_open_date
```

```
store_close_date
```

```
district
```

```
district_name
```

```
region
```

```
region_name
```


area
area_name
chain
chain_name
company
company_name
store_format
format_name
store_class

If %rowfound

 continue

else

 exit with failure, print sqlcode to error log

endif

while(1)

{

 Write store, store name, district, district name, region, region

 name, area, area name, chain, chain name, company, company name,

 store format, store format name, store class and store class

 (note store class has no description so the value is used as the description)

 to temporary master file (or ora string) and restart logic write records at max
 commit counter to final file

Write store, store open date, store close date to temporary attributes file (use
restart logic to write at max commit ctr).

fetch next record from cursor into local variables

```
if %rownotfound  
break;  
endif  
}
```

Declare an array to hold the store selling square footage.

Modify the driving cursor (C_org_info) to also fetch back the selling_square_ft column.

size_array()

Ensure that memory is allocated in the size arrays function for the new selling square footage area variable.

format_buffer()

Modify the ps_attr_buffer buffer so that the store selling square footage variable will come immediately after the close date variable.

table_process()

This function needs to be modified so that when the store, store open date and store close date are fetched, so is the store selling square footage.

write_file()

Zero_pad the selling square footage variable. Ensure that the variable is then written out to the attributes file.

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

Output file strmstr.dat

Store	Varchar2	24
Store Name	Varchar2	40
District	Varchar2	24
District Name	Varchar2	20
Region	Varchar2	24
Region Name	Varchar2	20
Area	Varchar2	24
Area Name	Varchar2	20
Chain	Varchar2	24
Chain Name	Varchar2	20
Company	Varchar2	24
Company Name	Varchar2	20
Store Format	Varchar2	24
Store Format Name	Varchar2	20
Store Class	Varchar2	24
Store Class Name	Varchar2	3-- description is same as class name

Output file strattr.dat

Store	Varchar2	24
Store Open Date	Date	8-- YYYYMMDD format
Store Close Date	Date	8-- YYYYMMDD format
Selling Square Footage	NUMBER	8

Fashion Sales Data Download [fsadnldf]

Design Overview

fsadnldf.pc is designed to extract sales data (current and historical) for all forecastable SKUs an output file. Forecastable SKUs are those with a forecast indicator equal to 'Y' on win_skus. These modules will only send records that meet the following criteria:

- New SKU record with sales (current week only)
- Existing SKU record that has been set to forecastable during a prior week (current week only)
- Existing SKU record set to forecastable during week (current and all historical records)
- Existing but reclassified SKU record (current and all historical records)

This module determine which sales records to send from the rag_skus_st_hist table by referencing the last sales export date field on therag_skus_st table. This field indicates the last end of week date that the sales record was sent to Retek Forecasting. As each logical unit of work is processed (SKU Store combination), the last end of week date unit (from system_variables) is updated into the last export date field.

Note: A SKU Store record will only be written under the following circumstances:

If the last sales export date is null (new SKU store record, existing SKU Store record set to forecastable, or a reclassification).

The last sales export date is less than or equal to the current week being processed.

Or the load sales indicator on the domain_dept, domain_class, or domain_subclass table has been set to 'Y' (indicating that the system administrator wishes to move a dept, class, or subclass, from one Retek Forecasting domain to another).

Once the extract files are written, they will be picked up by Retek Forecasting and loaded into their tables for processing.

Before processing any SKUs, the program must retrieve the domain aggregation level (department, class or subclass). Retek Forecasting has a database size which may necessitate the set up of multiple Retek Forecasting domains (database instances). The client DBA will need to determine at what level (department, class or subclass) this limit is not exceeded for SKU/store combinations. This level is maintained on system_options so that the correct cursor (dept, class, subclass) can be used during processing. Once this level is determined, the domain_matrix table maintains relationships between the chosen level (department, class or subclass) and the domain.

Scheduling Constraints

Processing Cycle: Weekly

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: By domain ID

Threading set-up: Since threads are determined by the value of the domain ID, the restart_program_status table should contain a row for each domain ID. The thread value of the domain ID should be used as the thread value on this table. The total number of domains/number of threads should be equal to the number of rows on the restart_program_status table. This value must be entered into the restart_control table num_threads field. Note that anytime a new domain is created that an additional row should be added to the restart_program_status table with the thread value equal to the domain ID and the restart_control table num_threads field must be incremented to equal the total number of domains.

Restart Recovery

The logical unit of work is sku and store for staple records. The commit (update rag_skus_st, move records from temporary output file to actual output file, truncate temporary file) should occur every 10,000 records.

Program Flow

N/A

Shared Modules

None.

Function Level Description

Init()

- Initialize restart variables.
- Collect system date and domain level information
- Setup output file.

Dept_cursor()

- If the domain level is 'D'ept this function will open, fetch, and close the dept level driving cursor. It will return a 1 when the cursor encounters a NO_DATA_FOUND.

Class_cursor()

- If the domain level is 'C'lass this function will open, fetch, and close the class level driving cursor. It will return a 1 when the cursor encounters a NO_DATA_FOUND.

Subclass_cursor()

- If the domain level is 'S'ubclass this function will open, fetch, and close the subclass level driving cursor. It will return a 1 when the cursor encounters a NO_DATA_FOUND.

Process()

- The function declares structures to hold data fetched from the driving cursor and for the array update of RAG_SKUS_ST.LAST_SALES_EXPORT_DATE.
- It calls the size_driv_cur_struct and size_update_sturct function to allocate memory for these structures.
- Depending on the domain level in will call dept_cursor, class_cursor, or subclass_cursor to open the needed driving cursor.
- It will then enter a while loop that terminated when one of the driving cursor function returns a 1. This loop will contain a call to dept_cursor, class_cursor, or subclass_cursor to fetch the needed driving cursor and a call to process_records to handle the records returned by the driving cursor.
- After the driving cursor loop terminates, it calls the post_store function to update any rag_skus_st records that didn't get processed by process_records.

Process_records()

- The function loops through the driving cursor array and processes records it contains.
 - If either the sku or the store changes, the store update array is filled with the previous sku/store rowid's on rag_skus_st. If the addition of this record fills the update array it will be posted to the database by calling the post_store function.
 - If the current counter is greater than the max counter, call to restart_commit will be made. A call to post_store is done before this to make sure any records in the output file are updated on rag_skus_st before a commit is performed.
 - Copy the current rowid, sku, and store into the previous holder variables.
 - Call write_fashion_file to write the current records information to the output file.
- If the driving cursor returned a one copy the previous rowid into the update array.

Post_store()

- This function will use the update array populated by process_records to update rag_skus_st.last_sales_export_date

Write_staple_file()

- This function writes the current record's information to the temporary output file. There is one output file for each given type of sales (regular, promotion, clearance)

Size_driv_cur_struct()

- Allocates memory for the driving cursor fetch array. It sizes the array to g_l_restart_max_counter.

Size_update_struct()

- Allocates memory for the win_wh update array. It sizes the array to g_l_restart_max_counter.

Final()

- Close restart recovery process
- Clean up output files.

I/O Specification

frsal01.dat – regular sales output file for thread one.

fpsal01.dat – promotional sales output file for thread one.

fcsal01.dat – clearance sales output file for thread one

Field Name	Field Type	Default Value	Description
WH	Char(20)		store identifier
SKU	Char(20)		RMS item identifier
End of week date	Char(8)		End of week date for record
Sales Quantity	Number(13)		Quantity of sales at wh for the sku in the week ending in with the end of week date. Decimal is included in output file.

Staple Sales Data Download [fsadnlds]

Design Overview

fsadnlds.pc is designed to extract sales data (current and historical) for all forecastable SKUs an output file. Forecastable SKUs are those with a forecast indicator equal to 'Y' on win_skus. These modules will only send records that meet the following criteria:

- New SKU record with sales (current week only)
- Existing SKU record that has been set to forecastable during a prior week (current week only)
- Existing SKU record set to forecastable during week (current and all historical records)
- Existing but reclassified SKU record (current and all historical records)

This module determine which sales records to send from the win_store_hist table by referencing the last sales export date field on the win_store table. This field indicates the last end of week date that the sales record was sent to Retek Forecasting. As each logical unit of work is processed (SKU Store combination), the last end of week date unit (from system_variables) is updated into the last export date field.

Note: That a SKU Store record will only be written under the following circumstances:

If the last sales export date is null (new SKU store record, existing SKU Store record set to forecastable, or a reclassification)

The last sales export date is less than or equal to the current week being processed.

Or the load sales indicator on the domain_dept, domain_class, or domain_subclass table has been set to 'Y' (indicating that the system administrator wishes to move a dept, class, or subclass, from one Retek Forecasting domain to another).

Once the extract files are written, they will be picked up by Retek Forecasting and loaded into their tables for processing.

Before processing any SKUs, the program must retrieve the domain aggregation level (department, class or subclass). Retek Forecasting has a database size which may necessitate the set up of multiple Retek Forecasting domains (database instances). The client DBA will need to determine at what level (department, class or subclass) this limit is not exceeded for SKU/store combinations. This level is maintained on system_options so that the correct cursor (dept, class, subclass) can be used during processing. Once this level is determined, the domain_matrix table maintains relationships between the chosen level (department, class or subclass) and the domain.

Scheduling Constraints

Processing Cycle:	Weekly
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	By domain ID

Threading set-up: Since threads are determined by the value of the domain ID, the restart_program_status table should contain a row for each domain ID. The thread value of the domain ID should be used as the thread value on this table. The total number of domains/number of threads should be equal to the number of rows on the restart_program_status table. This value must be entered into the restart_control table num_threads field. Note that anytime a new domain is created that an additional row should be added to the restart_program_status table with the thread value equal to the domain ID and the restart_control table num_threads field must be incremented to equal the total number of domains.

Restart Recovery

The logical unit of work is sku and store for staple records. The commit (update win_store, move records from temporary output file to actual output file, truncate temporary file) should occur every 10,000 records.

Program Flow

N/A

Shared Modules

None.

Function Level Description

Init()

- Initialize restart variables.
- Collect system date and domain level information
- Setup output file.

Dept_cursor()

- If the domain level is 'D'ept this function will open, fetch, and close the dept level driving cursor. It will return a 1 when the cursor encounters a NO_DATA_FOUND.

Class_cursor()

- If the domain level is 'C'lass this function will open, fetch, and close the class level driving cursor. It will return a 1 when the cursor encounters a NO_DATA_FOUND.

Subclass_cursor()

- If the domain level is 'S'ubclass this function will open, fetch, and close the subclass level driving cursor. It will return a 1 when the cursor encounters a NO_DATA_FOUND.

Process()

- The function declares structures to hold data fetched from the driving cursor and for the array update of WIN_STORE.LAST_SALES_EXPORT_DATE.
- It calls the size_driv_cur_struct and size_update_sturct function to allocate memory for these structures.
- Depending on the domain level in will call dept_cursor, class_cursor, or subclass_cursor to open the needed driving cursor.
- It will then enter a while loop that terminated when one of the driving cursor function returns a 1. This loop will contain a call to dept_cursor, class_cursor, or subclass_cursor to fetch the needed driving cursor and a call to process_records to handle the records returned by the driving cursor.
- After the driving cursor loop terminates, it calls the post_store function to update any win_store records that didn't get processed by process_records.

Process_records()

- The function loops through the driving cursor array and processes records it contains.
 - If either the sku or the store changes, the store update array is filled with the previous sku/store rowid's on win_store. If the addition of this record fills the update array it will be posted to the database by calling the post_store function.
 - If the current counter is greater than the max counter, call to restart_commit will be made. A call to post_store is done before this to make sure any records in the output file are updated on win_store before a commit is performed.
 - Copy the current rowid, sku, and store into the previous holder variables.
 - Call write_staple_file to write the current records information to the output file.
- If the driving cursor returned a one copy the previous rowid into the update array.

Post_store()

- This function will use the update array populated by process_records to update win_store.last_sales_export_date

Write_staple_file()

- This function writes the current record's information to the temporary output file. There is one output file for each given type of sales (regular, promotion, clearance)

Size_driv_cur_struct()

- Allocates memory for the driving cursor fetch array. It sizes the array to g_l_restart_max_counter.

Size_update_struct()

- Allocates memory for the win_wh update array. It sizes the array to g_l_restart_max_counter.

Final()

- Close restart recovery process
- Clean up output files.

I/O Specification

Srsal01.dat – regular sales output file for thread one.

Spsal01.dat – promotional sales output file for thread one.

Scsal01.dat – clearance sales output file for thread one

Field Name	Field Type	Default Value	Description
WH	Char(20)		store identifier
SKU	Char(20)		RMS item identifier
End of week date	Char(8)		End of week date for record
Sales Quantity	Number(13)		Quantity of sales at wh for the sku in the week ending in with the end of week date. Decimal is included in output file.

[ftmednld.pc]

Design Overview

Currently, no extracts exist for the time dimension. So as not to have to maintain the calendar in multiple places (i.e. RMS, RDF and RPP), a time dimension extract is required that will download the RMS calendar, including the following fields: year, half, quarter, month, week, day and date (in a yyyyymmdd format). The downloaded information would only use the 454 calendar format. The download would include the entire calendar in the RMS. The extract must account for a fiscal year that could be different than the standard year in the calendar table. A field on the System Options table indicates the month in which the fiscal year begins. For example, if the fiscal year begins on the 3rd month, the following chart highlights how this impacts the extract (the following is a subset of the data on the Calendar table):

First Day	Year	Month	# of Weeks in Month
25-OCT-99	1999	11	4
22-NOV-99	1999	12	5
27-DEC-99	2000	1	4
24-JAN-00	2000	2	4
21-FEB-00	2000	3	5
27-MAR-00	2000	4	4

If the fiscal year followed standard calendar, the first day of the year 2000 would be 27-Dec-99. However, for the fiscal year which starts on the 3rd month, the first day of the year 2000 would be 21-FEB-00. Therefore, 20-FEB-00 would be extracted as 1999 (year), 2 (half), 4 (quarter), 12 (month), 4 (week), 7 (day), 20000220 (date). If the year followed the regular calendar, 20-FEB-00 would be extracted as 2000 (year), 1 (half), 1 (quarter), 2 (month), 4 (week), 7 (day), 20000220 (date).

Table-To-File'

This program fetches the earliest and latest dates from the calendar also incorporating the start_of_half_month from the system_options table when fetching the earliest date.

Driving Cursor:

NA

Output Specifications

The file outputted will be named `ftmehier.dat`.

Output File Format:

Field Name	Field Type	Description
Year	Char(4)	The 454 year
Half	Char(1)	The 454 half of the year, valid values are 1 or 2
Quarter	Char(1)	The 454 quarter of the year, valid values 1-4
Month	Char(2)	The 454 month of the year, valid values 1-12
Week	Char(2)	The 454 week of the year, valid values 1-53
Day	Char(1)	The 454 day of the current week, valid values 1-7
Date	Char(8)	The date from which the 454 data was derived, in YYYYMMDD format

Function Level Description

`main()`:

The standard Retek `main()` function. Calls `init()`, `process()`, and `final()`.

`init()`:

Initialize restart recovery by calling `retex_init()` and set up the output file.

`format_buffer()`:

Formats the string that will be used to write to the output file.

`get_dates()`:

Mutates output arguments with first and last calendar date fetched from the `calendar.first_day` field. First calendar date is determined by taking the first record from the calendar table ordered in chronological order with respect to `first_date`. The fetched record's `month_454` field must match the absolute value of the `system_options.start_of_half_month` field. Last calendar date is determined by fetching the first record while the calendar table records are ordered in a descending order with respect to the `first_day` field.

`increment_date()`

Mutates input/output arguments to hold incremented date.

increment_454()

Mutates input/output arguments to hold incremented 454 date. Also fetches the calendar.no_of_weeks field from the calendar table's row whose first day field corresponds to the present date.

To determine the 454 day, month, weeks, yearly weeks, quarter, half and year, simply increment their current values by one if the corresponding date counters justify the incrementation. If any one of them turns over reset them to 1. Note that weeks turn over when the no_of_weeks value fetched from the calendar table is no longer greater than or equal to the current week value.

init_454()

Initializes argument 454 date instance's date fields.

write_fdetl()

Writes data from argument to output file.

process():

This function first makes a call to format_buffer(). Then allocates a date struct to hold the 454 date. It then calls the init_454() function with a pointer to the 454 date struct as argument. It then fetches the earliest and latest values of calendar.first_day into local variables by calling get_dates(). For each day in the date range (including the earliest and latest dates), the current calendar and 454 dates are calculated by calling increment_date() and increment_454().

A record containing all of the 454 values for the date, in addition to the date itself (in YYYYMMDD format) will then be written to the file by calling write_fdetl().

final():

Take care of file clean up and complete the restart recovery process by calling retek_close().

Scheduling Considerations

This program can be run ad hoc.

Restart/Recovery

Due to the relatively small amount of processing this program performs, restart recovery will not be used. The calls to retek_init() and retek_close() are used in the program only for logging purposes (to prevent double-runs).

Forecast Warehouse Download [fwhdnld]

Design Overview

fwhdnld.pc is designed to download RMS warehouses, including the warehouse description to an output file to be picked up by Retek Demand Forecasting (RDF) and loaded into their tables for processing.

Retek Forecasting has a database size which may necessitate the set up of multiple Retek Forecasting domains (database instances). Fortunately, the segmentation level is held in the merchandise, not product hierarchy. Therefore, only one hierarchy file will be created and RDF will load a copy of this file into all domains.

This program can be run ad-hoc, whenever it is necessary to download any warehouse changes to RDF.

Scheduling Constraints

Processing Cycle:	Ad-hoc
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	Due to the limited size of the warehouse table, no threading is required.

Restart Recovery

The logical unit of work is warehouse. Due to the limited size of the Warehouse table, the commit can occur after all warehouses have been processed (i.e. one commit point).

The driving cursor should select all warehouses and their descriptions from the Warehouse table. All single and double quotes should be replaced with a space.

Program Flow

Init()

Performs standard initialization processing.

Process()

Using the driving cursor, fetch every warehouse in the Warehouse table. For each warehouse call write_file() to write the information to the output file.

Write_file()

Write the warehouse ID and description to the output file.

Final()

Standard batch wrap-up processing.

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

Output file whmstr.dat

Example:

01234567890123456789012345678901234567890123456789

0001 DC Central

Field	Data Type	Description
Warehouse	Varchar2(20)	RMS Warehouse ID – left-justified, padded with zeros up to four places, right-padded remaining 16 places with spaces.
Description	Varchar2(4)	Warehouse description – left-justified, right-padded with spaces.

Gcdupld.pc

Design Overview

The batch program will provide the ability to upload geocodes from an outside source into the RMS tables. The program will allow the deletion of current geocodes and the creation of new geocodes in the system. A flat file will be used to feed the program the additions and deletions to the geocode tables. Validation will occur to determine if duplicate records exist, dependencies exist, and the flat file is in the correct format. If errors occur in the validation of the record, it will be written out to a reject file to allow further investigation of the record.

Assumption: The items that are passed in are only the ones that need to be added or deleted from the system.

Scheduling Constraints

Processing Cycle: ad hoc

Scheduling Diagram: ad hoc

Pre-Processing: none

Post-Processing: none

Threading Scheme:

Program Flow

Main () –

This function will call the appropriate functions within the program

- Init ()
- Process()
- File Process ()
- Final()

Init()

- This function will initialize restart recovery for the program.
- Open a temporary output file in write mode. This file will be used to create the reject file at the end of the program.
- Read in the information that is placed on the flat file. This should be done using the get_record command.

Size_arrays()

- This function will size all of the various structures that will be used to insert or delete records from the geocode tables .
- All of the counters that are declared for the array inserts and deletes should be initialized.

Process()

This function drives the population of the geocode temp table. This temp table will be used in the program to facilitate what order the geocodes will be inserted or deleted from the system.

- The input file will have a THEAD line for each geocode in the system. Create a while loop to continue to loop through all of the THEAD lines until the FTAIL is reached in the input file.
- Once a THEAD line is read into the program call the Validate _THEAD function to validate that the data being received into the system is valid.
- After the validation of the item has occurred then call the function insert_temp_records(). This function will insert a new record into the geocode_temp table. The temp table will be used in later functions to determine what geocodes already exist in the system and what geocodes need to be inserted or deleted.

Validate THEAD()-

This function will place all of the values in the THEAD line into local variables such that they can be used throughout the rest of the program.

Insert geocode_temp ()

This function will insert a new record into one of the new temp table. The values that will be inserted are geocode, geocode description, geocode level, and the add/delete indicator.

File_process ()

This function will call the appropriate functions that will determine what geocodes are to be added or deleted from the various geocode hierarchy tables.

The driving cursor for this function will select the geocode, geocode description, level, and the add/delete indicator from the temp table ordered by level, geocode, and the add/delete indicator.

If the add/delete indicator is set to 'A'

- Call the function add_country_geocode
- Call the function add_state_geocode
- Call the function add_county_geocode
- Call the function add_city_geocode
- Call the function add_district_geocode
- After all of the insert arrays are filled the function insert_geocode() will be called to insert the arrays into the appropriate tables.

If the Flag is set to 'D' then

- Call the function delete_district_geocode
- Call the function delete_city_geocode
- Call the function delete_county_geocode
- Call the function delete_state_geocode
- Call the function delete_country_geocode
- After all of the delete arrays are filled the function delete_geocode() will be called to delete the arrays from the appropriate tables.

Add_district_geocode()

- This function will insert a new district geocode into the district geocode table. Validate that all of the values that are being inserted for levels above are valid in the system. The country, state, county, and city must all be valid and either exist on the district_geocode, city_geocode, county_geocode, state_geocode, country_geocode tables or on the geocode_temp table as an add. If one of the values does not exist then the record would not be valid and it should be written out the reject file.

Add_city_geocode()

- This function will insert a new city geocode into the city geocode table. Validate that all of the values that are being inserted for levels above are valid in the system. The country, state, and county must all be valid and either exist on the city_geocode, county_geocode, state_geocode, country_geocode tables or on the geocode_temp table as an add. If one of the values does not exist then the record would not be valid and it should be written out the reject file.

Add_county_geocode()

- This function will insert a new county geocode into the county geocode table. Validate that all of the values that are being inserted for levels above are valid in the system. The country, and state must all be valid and either exist on the county_geocode, state_geocode, country_geocode tables or on the geocode_temp table as an add. If one of the values does not exist then the record would not be valid and it should be written out the reject file.

Add_state_geocode()

- This function will insert a new state geocode into the state geocode table. Validate that all of the values that are being inserted for levels above are valid in the system. The country must be valid and either exist on the state_geocode, country_geocode tables or on the geocode_temp table as an add. If one of the values does not exist then the record would not be valid and it should be written out the reject file.

Add_country_geocode()

- This function will insert a new country geocode into the country geocode table. Validate that the country does not exist in the system. If it does then write it out the reject file otherwise add the country geocode into the system.

Insert geocode ()

All of the add functions listed above should fill the following insert arrays for each of the levels: insert_country_geocode, insert_state_geocode, insert_county_geocode, insert_city_geocode, insert_district_geocode. Once all of the arrays are filled this function will insert them into the appropriate tables. Since there are dependencies across the hierarchy the country geocodes must be inserted first, continuing down the hierarchy until the district geocodes are inserted.

Delete_district_geocode()

This function will fill the district delete array with values that need to be removed from the system. District geocodes can always be deleted since they do not have any dependencies.

Delete_city_geocode()

This function will fill the city delete array with values that need to be removed from the system. Before the records can be deleted from the system a check must be done to see if this value exists on any of the other levels of the geocode hierarchy. If the city exists on any of the following tables it cannot be deleted from the system: geocode_temp (Add), and district_geocodes. If a dependency is found then the record must be written out to the reject file. If no dependencies are found then the value may be deleted from the system.

Delete_county_geocode()

This function will fill the county delete array with values that need to be removed from the system. Before the records can be deleted from the system a check must be done to see if this value exists on any of the other levels of the geocode hierarchy. If the county exists on any of the following tables it cannot be deleted from the system: geocode_temp (Add), district_geocodes, and city_geocodes. If a dependency is found then the record must be written out to the reject file. If no dependencies are found then the value may be deleted from the system.

Delete_state_geocode()

This function will fill the state delete array with values that need to be removed from the system. Before the records can be deleted from the system a check must be done to see if this value exists on any of the other levels of the geocode hierarchy. If the state exists on any of the following tables it cannot be deleted from the system: geocode_temp (Add), district_geocodes, city_geocodes, and county_geocodes. If a dependency is found then the record must be written out to the reject file. If no dependencies are found then the value may be deleted from the system.

Delete_country_geocode()

This function will fill the country delete array with values that need to be removed from the system. Before the records can be deleted from the system a check must be done to see if this value exists on any of the other levels of the geocode hierarchy. If the country exists on any of the following tables it cannot be deleted from the system: geocode_temp (Add), district_geocodes, city_geocodes, county_geocodes, and state_geocodes. If a dependency is found then the record must be written out to the reject file. If no dependencies are found then the value may be deleted from the system.

Delete_geocode ()

All of the delete functions listed above should fill the following delete arrays for the various levels: delete_country_geocode, delete_state_geocode, delete_county_geocode, delete_city_geocode, delete_district_geocode. Once all of the arrays are filled this function will delete them into the appropriate tables. Since there are dependencies across the hierarchy the district geocodes must be deleted first, continuing up the hierarchy until the country geocodes are deleted.

Final ()

This function will close all open files and delete all of the information off of the temp files.

I/O Specification

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	File head descriptor	Char(5)	FHEAD	Describes file line type
	Line id	Char(10)	0000000001	Sequential file line number
	Gentran ID	Char(5)	'GCUP'	Identifies which translation Gentran uses
	Current date	Char(14)		File date in YYYYMMDDHH24 MISS format
FDETL	File record descriptor	Char(5)	THEAD	Start of an invoice transaction
	Line id	Char(10)		Sequential file line number
	Country Geocode	Char(4)		Country Geocode
	State Geocode	Char(4)		State Geocode
	County Geocode	Char(4)		County Geocode
	City Geocode	Char(4)		City Geocode
	District Geocode	Char(4)		District Geocode
	Geocode Description	Char(40)		Geocode Description
	Add Delete Ind	Char(1)		Add/delete Indicator
FTAIL	File record descriptor	Char(5)	FTAIL	Marks end of file
	Line id	Char(10)		Sequential file line number
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

[posgpdlld.pc]

Design Overview

Occasionally, retailers need to perform department level sales for merchandise. To accommodate this, a new download program (posgpdlld.pc) is created to download group number/department information to individual Point-of-Sale (POS) systems.

Assumption: The dataset that defines group numbers and departments is small. With this assumption, a full extract of RMS group number/department combinations occurs for every run of the batch program.

Scheduling Constraints

Processing Cycle:	ad hoc
Scheduling Diagram:	Run after reclsdly.pc
Pre-Processing:	reclsdly.pc
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

Logical Unit of Work

Group_no/dept

Driving Cursor

```
SELECT group_no,
       dept,
       ','||to_char(group_no)||';'||to_char(dept)
FROM deps
WHERE (group_no > NVL(:ps_restart_group_no, -999)
      OR (group_no = :ps_restart_group_no
          AND dept > :ps_restart_dept))
ORDER BY group_no, dept;
```

Shared Modules

N/A

Function Level Description

Global structure of arrays for data processing:

```
struct fetch_rec
{
    char (*as_group_no)[NULL_GROUP_NO];
    char (*as_dept)[NULL_DEPT];
    char (*as_restart_string)[NULL_RESTART]
};
```

Called from init():

int get_date()

Fetches the vdate (business date) from the period table and places it into a program global variable.

int size_array(struct fetch_recs *ioa_data_buffer)

Sizes the arrays used in database FETCHES by using the calloc function.

void format_strings()

Creates format strings used in file outputs.

int init()

Performs standard Retek restart/recovery initialization. All file handling occurs here, such as declaration of a temp file, and opening of files. The FHEAD line is also written to the file, in the case of a fresh start.

int process()

The driving cursor is used to fetch records from the database. The group_no and dept are written to the temp file using the FDETL format string defined previously. Once all data has been fetched an FTAIL line is written to the file.

int final()

Performs closing operations. A call to restart_close() is made, and the temp file is removed from the system.

I/O Specification

Record Type	Field	Datatype	Valid Values	Description
FHEAD	Record Type	Char(5)	'FHEAD'	Record Identifier
	Line ID	Number(10)	0000000001	Line Sequence Identifier
	File Name	Char(4)	'GRPD'	File Identifier
	File Date	Char(14)		Date the file was created in 'YYYYMMDDHHMMSS' format
FDETL	Record Type	Char(5)	'FDETL'	Record Identifier
	Line ID	Number(10)		Line Sequence Identifier
	Group no	Numeric ID (4) (left-justified character field)		Valid RMS Group Number.
	Dept	Numeric ID (4) (left-justified character field)		Valid RMS Department for the Group Number.
FTAIL	Record Type	Char(5)	'FTAIL'	Record Identifier
	Line ID	Number(10)		Line Sequence Identifier
	Line Total	Number(10)		Total number of FDETL lines in the file.

Sales History Rollup by Department, Class, and Subclass [hstbld]

Design Overview

The sales history rollup routine will extract sales history information for each SKU from the rag_skus_st_hist and win_store_hist tables. The history information will be rolled up to the subclass, class, and dept level to be written to: dept_sales_hist, class_sales_hist, and subclass_sales_hist.

For each SKU, data to be saved includes sales qty, value, gross profit, and sales rate. This data must be collected from several tables including win_store_hist, win_skus, rag_skus_st_hist, rag_skus, and rag_style. Letting the database (server) roll up the totals verse using a loop on the client enhances speed. Using a VIEW that contains all SKU information for the current week enhances simplicity. Data can then be summed from this single view instead of having to join across all 5 tables in a single select statement.

The rebuild program can be run in one of two ways:

First, if the program is run with a run-time parameter of 'rebuild', the program will read data (dept, class, and subclass) off the manually input mask_rebuild table, which will determine what is rebuilt. This process is used after items are reclassified from one merchandise hierarchy to another. Rebuilding a department will rebuild each class and subclass within the department, thus, only one row is required on mask rebuild for the department. This type of rebuilding process will rebuild data from all dates on the win_store_hist and rag_skus_st_hist tables, rolling them to the department, class, and subclass level.

Second, if the program is run with a run-time parameter of 'weekly', the program will build sales information for all dept/class/subclass combinations only for the current end of week date.

mask_rebuild table:

DEPT	CLASS	SUBCLASS	
X	NULL	NULL	Rebuild Department
X	X	NULL	Rebuild Class
X	X	X	Rebuild Subclass

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DEPT_SALES_HIST	No	Yes	Yes	Yes	No
CLASS_SALES_HIST	No	Yes	Yes	Yes	No
SUBCLASS_SALES_HIST	No	Yes	Yes	Yes	No
V_REBUILD_SALES_HIST	No	Yes	No	No	No
V_SALES_HIST	No	Yes	No	No	No
PERIOD	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE 3 (weekly)
PHASE AD-HOC (weekly)

Scheduling Diagram: Must run after complete weekly sales have been updated by posupld. Also should be re-run on demand when a sales rollup request has been given for a given dept, class or subclass

Pre-Processing: N/A

Post-Processing: hstbld_post()
Truncates the mask rebuild table.

Threading Scheme: N/A (single threaded)

Restart Recovery

If program is run for reclassifying SKUs (second run time parameter = 'rebuild'), the driving cursor for the program will be:

```
SELECT vrs.dept,
       vrs.class,
       vrs.subclass,
       vrs.store,
       to_char(vrs.eow_date,'YYYYMMDDHH24MISS'),
       vrs.week_454,
       vrs.month_454,
       vrs.year_454,
       vrs.sales_type,
       SUM(NVL(vrs.sales,0)),
       SUM(NVL(vrs.value,0)),
       SUM(NVL(vrs.gp,0)),
       SUM(NVL(vrs.forecast_sales,0))
FROM v_rebuild_sales_hist vrs,
     mask_rebuild mr
WHERE ((vrs.dept = mr.dept
       AND vrs.class = mr.class
       AND vrs.subclass = mr.subclass
       AND mr.class is not NULL
       AND mr.subclass is not NULL)
       OR (vrs.dept = mr.dept
          AND vrs.class = mr.class
          AND mr.class is not NULL
```

```
AND mr.subclass is NULL)
OR (vrs.dept = mr.dept
AND mr.class is NULL
AND mr.subclass is NULL))
AND (vrs.dept > NVL(:ov_restart_dept, -999) OR
(vrs.dept = :ov_restart_dept AND
(NVL(vrs.class,0) > :ov_restart_class OR
(NVL(vrs.class,0) = :ov_restart_class AND
(NVL(vrs.subclass,0) > :ov_restart_subclass)
)
)
)
)
GROUP BY vrs.dept,
vrs.class,
vrs.subclass,
vrs.store,
vrs.eow_date,
vrs.week_454,
vrs.month_454,
vrs.year_454,
vrs.sales_type
ORDER BY vrs.dept,
vrs.class,
vrs.store,
vrs.eow_date;
```

If program is run for current end of week (second run time parameter = 'weekly'), the driving cursor for the program will be:

```
SELECT vsh.dept,
       vsh.class,
       vsh.subclass,
       vsh.store,
       to_char(vsh.eow_date, 'YYYYMMDDHH24MISS'),
       vsh.week_454,
       vsh.month_454,
       vsh.year_454,
       vsh.sales_type,
       SUM(NVL(vsh.sales,0)),
       SUM(NVL(vsh.value,0)),
       SUM(NVL(vsh.gp,0)),
       SUM(NVL(vsh.forecast_sales,0))
FROM v_sales_hist vsh
WHERE (vsh.dept > NVL(:ov_restart_dept, -999) OR
       (vsh.dept = :ov_restart_dept AND
        (NVL(vsh.class,0) > :ov_restart_class OR
         (NVL(vsh.class,0) = :ov_restart_class AND
          (NVL(vsh.subclass,0) > :ov_restart_subclass)
         )
        )
       )
GROUP BY vsh.dept,
         vsh.class,
```



```
vsh.subclass,  
vsh.store,  
vsh.eow_date,  
vsh.week_454,  
vsh.month_454,  
vsh.year_454,  
vsh.sales_type  
ORDER BY vsh.dept,  
vsh.class,  
vsh.store,  
vsh.eow_date;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

init()

- Initialize restart recovery
- If processing current end of week, call check_eow_date()
- Initialize structure arrays for subclass_sales_hist insert, subclass_sales_hist update, class_sales_hist insert, class_sales_hist update, and sales_history fetch.
- Check_eow_date()
- Check that vdate is a valid end of week date

process()

- Open driving cursor
- Array fetch cursor
- Loop through array to process records.
- Increment dept and class sales variables for running totals of sales, value, gp, and forecast_sales.
- Call process_subclass().
- If dept/class/store/eow_date/sales_type changes, call process_class().
- If dept/store/eow_date/sales_type changes, call process_dept().
- Call process_subclass(), process_class(), and process_subclass() to process last record fetched (last record is not processed within above loop).
- Call insert_class()
- Call update_class()
- Call insert_subclass()
- Call update_subclass

Process_subclass()

- Fetch previous sales, value, gp, and forecast_sales value from subclass_sales_hist
- If fetch is not found (no record exists), add values to subclass_insert array
- If fetch is found,
 - Add values to subclass_update array
 - Increment delta variables to hold running totals for difference in subclass' sales and updated sales, value and updated value, gp and updated gp, forecast_sales and updated forecast_sales.

Process_class()

- Fetch rowid from class_sales_hist
- If fetch is not found, add values to class_insert array
- If fetch is found,

- Add values to class_update array
- Increment delta variables to hold running totals for difference in class' sales, value, gp, and forecast_sales by adding deltas from subclass' sales, value, gp, and forecast_sales.
- Reset subclass delta variables and class running total variables.

Process_dept()

- Perform update of dept_sales_hist
- If update is not found (record does not exist), perform insert into dept_sales_hist
- Reset class delta variables and dept running total variables.

Insert_class()

- Perform array insert of class_sales_hist.

Update_class()

- Perform array update of class_sales_hist.

Insert_subclass()

- Perform array insert of subclass_sales_hist.

Update_subclass()

- Perform array update of subclass_sales_hist.

I/O Specification

N/A

Technical Issues

N/A

Purge Sales History [hstprg]

Design Overview

Deletes from RAG_SKUS_ST_HIST and RAG_SKUS_WH_HIST where data is more than a specified number of months old. Number of months for retention of fashion style history is specified by unit_options.RAG_SKUS_HISTORY_MONTHS.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
RAG_SKU_ST_HIST	No	No	No	No	Yes
RAG_SKUS_WH_HIST	No	No	No	No	Yes
SYSTEM_VARIABLES	No	Yes	No	No	No
UNIT_OPTIONS	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE AD-HOC (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Weekly Stock on Hand and Retail Value Update for Fashion SKU/Store [hstwkfst]

Design Overview

Run weekly to update the current stock on hand, current retail and average cost values for each sku on RAG_SKUS_ST_HIST.

If a record does not exist on win_store_hist AND the stock_on_hand is greater than 0 AND the history level indicator is A (see below) then the record is written out to an ascii file. All threads will write to the same file (via the restart file write function) which will be loaded into the win_store_hist table by the SQL-Loader utility.

Possible values for the inventory history level indicator:

N - No history information is maintained

I - Information is only maintained on items sold within the week

A - Information is maintained on all items with a stock on hand at the end of the week greater than zero, regardless of sales

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No
WIN_STORE	Yes	Yes	No	No	No
WIN_STORE_HIST	Yes	No	Yes	Yes	No

Scheduling Constraints

Processing Cycle: PHASE 3 (weekly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: STORE
V_restart_store

Restart Recovery

History Level 'A' Driving Query

```
SELECT /*+ INDEX(win_store pk_win_store) */
      ws.sku,
      ws.store,
      NVL(ws.stock_on_hand,0),
      NVL(ws.unit_retail,0),
      NVL(ws.av_cost,0)
FROM win_store ws,
      v_restart_store rv
WHERE ws.sku          = ws.sku
      AND ws.store      = ws.store
      AND rv.driver_value = ws.store
      AND rv.driver_name  = :ora_restart_driver_name
      AND rv.num_threads  = :ora_restart_num_threads
      AND rv.thread_val   = :ora_restart_thread_val
      AND (ws.sku > NVL(:ora_restart_sku,-999) OR
           (ws.sku = :ora_restart_sku AND
            ws.store > :ora_restart_store))
ORDER BY ws.sku,
         ws.store;
```

History Level 'I' Driving Query

```
SELECT /*+ INDEX(win_store_hist pk_win_store_hist) */
      wsh.sku,
      wsh.store,
      ROWIDTOCHAR(wsh.rowid)
```

```
FROM win_store_hist wsh,
     v_restart_store rv
WHERE wsh.sales_type = 'R'
     AND wsh.eow_date = TO_DATE(:current_eow)
           AND wsh.sku = wsh.sku
           AND wsh.store = wsh.store
           AND wsh.sales_type = wsh.sales_type
           AND wsh.eow_date = wsh.eow_date
     AND rv.driver_value = wsh.store
     AND rv.driver_name = :ora_restart_driver_name
     AND rv.num_threads = :ora_restart_num_threads
     AND rv.thread_val = :ora_restart_thread_val
     AND (wsh.sku > NVL(:ora_restart_sku,-999) OR
          (wsh.sku = :ora_restart_sku AND
           wsh.store > :ora_restart_store))
ORDER BY wsh.sku,
         wsh.store;
```


Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Weekly Stock on Hand and Retail Value Update for Fashion SKU/Warehouse [hstwkfwh]

Design Overview

Run weekly to update the current stock on hand, retail values and average cost for each sku on RAG_SKUS_WH_HIST. The program must be run on the last day of the week as scheduled.

Possible values for the inventory history level indicator:

- N - No history information is maintained
- I - Information is only maintained on items sold within the week
- A - Information is maintained on all items with a stock on hand at the end of the week greater than zero, regardless of sales

Scheduling Constraints

Processing Cycle: PHASE 3 (weekly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: WH
V_restart_wh

Restart Recovery

History Level 'A' Driving query

```
SELECT /*+ INDEX(rag_skus_wh pk_rag_skus_wh) */  
    rsw.sku,  
    rsw.wh,  
    NVL(rsw.stock_on_hand,0),  
    NVL(rsw.unit_retail,0),  
    NVL(rsw.av_cost,0)  
FROM rag_skus_wh rsw,
```

```

        v_restart_wh rv
WHERE rsw.sku      = rsw.sku

AND rsw.wh        = rsw.wh

AND rv.driver_value = rsw.wh

AND rv.driver_name = :ora_restart_driver_name

AND rv.num_threads = :ora_restart_num_threads

AND rv.thread_val  = :ora_restart_thread_val

AND (rsw.sku > NVL(:ora_restart_sku,-999) OR

     (rsw.sku = :ora_restart_sku AND

      rsw.wh > :ora_restart_wh))

ORDER BY rsw.sku,

        rsw.wh;

```

History Level 'I' Driving query

```

SELECT /*+ INDEX(rag_skus_wh_hist pk_rag_skus_wh_hist) */

        rswh.sku,

        rswh.wh,

        ROWIDTOCHAR(rswh.rowid)

FROM rag_skus_wh_hist rswh,

        v_restart_wh rv

WHERE rswh.eow_date = TO_DATE(:current_eow)

AND rswh.sku      = rswh.sku

AND rswh.wh        = rswh.wh

AND rswh.eow_date  = rswh.eow_date

AND rv.driver_value = rswh.wh

AND rv.driver_name = :ora_restart_driver_name

AND rv.num_threads = :ora_restart_num_threads

```

```
AND rv.thread_val = :ora_restart_thread_val
AND (rsw.skus > NVL(:ora_restart_sku,-999) OR
(rsw.skus = :ora_restart_sku AND
rsw.wh > :ora_restart_wh))
ORDER BY rsw.skus,
rsw.wh;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Weekly Stock On Hand and Retail Value Update for Staple SKU/Store [hstwksst]

Design Overview

Run weekly to update the current stock on hand and retail values for each sku on WIN_STORE_HIST.

If a record does not exist on win_store_hist AND the stock_on_hand is greater than 0 AND the history level indicator is A (see below) then the record is written out to an ascii file. All threads will write to the same file (via the restart file write function) which will be loaded into the win_store_hist table by the SQL-Loader utility.

Possible values for the inventory history level indicator:

N - No history information is maintained

I - Information is only maintained on items sold within the week

A - Information is maintained on all items with a stock on hand at the end of the week greater than zero, regardless of sales

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No
WIN_STORE	Yes	Yes	No	No	No
WIN_STORE_HIST	Yes	No	Yes	Yes	No

Scheduling Constraints

Processing Cycle: PHASE 3 (weekly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: STORE
V_restart_store

Restart Recovery

History Level 'A' Driving Query

```
SELECT /*+ INDEX(win_store pk_win_store) */
      ws.sku,
      ws.store,
      NVL(ws.stock_on_hand,0),
      NVL(ws.unit_retail,0),
      NVL(ws.av_cost,0)
FROM win_store ws,
      v_restart_store rv
WHERE ws.sku          = ws.sku
      AND ws.store      = ws.store
      AND rv.driver_value = ws.store
      AND rv.driver_name  = :ora_restart_driver_name
      AND rv.num_threads  = :ora_restart_num_threads
      AND rv.thread_val   = :ora_restart_thread_val
      AND (ws.sku > NVL(:ora_restart_sku,-999) OR
           (ws.sku = :ora_restart_sku AND
            ws.store > :ora_restart_store))
ORDER BY ws.sku,
         ws.store;
```

History Level 'I' Driving Query

```
SELECT /*+ INDEX(win_store_hist pk_win_store_hist) */
      wsh.sku,
      wsh.store,
      ROWIDTOCHAR(wsh.rowid)
```

```
FROM win_store_hist wsh,  
     v_restart_store rv  
WHERE wsh.sales_type = 'R'  
      AND wsh.eow_date = TO_DATE(:current_eow)  
      AND wsh.sku      = wsh.sku  
      AND wsh.store    = wsh.store  
      AND wsh.sales_type = wsh.sales_type  
      AND wsh.eow_date = wsh.eow_date  
      AND rv.driver_value = wsh.store  
      AND rv.driver_name = :ora_restart_driver_name  
      AND rv.num_threads = :ora_restart_num_threads  
      AND rv.thread_val  = :ora_restart_thread_val  
      AND (wsh.sku > NVL(:ora_restart_sku,-999) OR  
           (wsh.sku = :ora_restart_sku AND  
            wsh.store > :ora_restart_store))  
ORDER BY wsh.sku,  
         wsh.store;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Weekly Stock on Hand and Retail Value Update for Staple SKU/Warehouse [hstwkswh]

Design Overview

Run weekly to update the current stock on hand and retail values for each SKU on WIN_WH_HIST. The program must be run on the last day of the week as scheduled.

Possible values for the inventory history level indicator:

- N - No history information is maintained
- I - Information is only maintained on items sold within the week
- A - Information is maintained on all items with a stock on hand at the end of the week greater than zero, regardless of sales

Scheduling Constraints

Processing Cycle:	PHASE 3 (weekly)
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	WH V_restart_wh

Restart Recovery

History Level 'A' Driving query

```
SELECT /*+ INDEX(win_wh pk_win_wh) */
      ww.sku,
      ww.wh,
      NVL(ww.stock_on_hand,0),
      NVL(ww.unit_retail,0),
      NVL(ww.av_cost,0)
FROM win_wh ww,
```

```
        v_restart_wh rv
WHERE ww.sku      = ww.sku

AND ww.wh        = ww.wh

AND rv.driver_value = ww.wh

AND rv.driver_name = :ora_restart_driver_name

AND rv.num_threads = :ora_restart_num_threads

AND rv.thread_val = :ora_restart_thread_val

AND (ww.sku > NVL(:ora_restart_sku,-999) OR

     (ww.sku = :ora_restart_sku AND

      ww.wh > :ora_restart_wh))

ORDER BY ww.sku,

        ww.wh;
```

History Level 'I' Driving query

```
SELECT /*+ INDEX(win_wh_hist pk_win_wh_hist) */

        ww.sku,

        ww.wh,

        ROWIDTOCHAR(ww.rowid)

FROM win_wh_hist ww,

        v_restart_wh rv

WHERE ww.eow_date = TO_DATE(:current_eow)

AND ww.sku      = ww.sku

AND ww.wh        = ww.wh

AND ww.eow_date = ww.eow_date

AND rv.driver_value = ww.wh

AND rv.driver_name = :ora_restart_driver_name

AND rv.num_threads = :ora_restart_num_threads
```

```
AND rv.thread_val = :ora_restart_thread_val
AND (wwh.sku > NVL(:ora_restart_sku,-999) OR
(wwh.sku = :ora_restart_sku AND
wwh.wh > :ora_restart_wh))
ORDER BY wwh.sku,
wwh.wh;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Upload Customs tariff files [htsupld]

Design Overview

This batch program will be run whenever an updated US customs tariff file is available (probably twice a year) to upload HTS tariff information from the file into RMS HTS tables. The program will handle both the initial HTS information load as well as mid-year HTS updates that are supplied by the US government. The initial upload is handled by inserting information from the file into the tables; updating information already in the tables is handled by adjusting the effective dates of the existing HTS records and inserting a new set of HTS records into the tables.

Updating HTS records should follow the following guidelines:

- No HTS records with the same HTS and import country should have overlapping effect_from and effect_to dates. Import country is passed as an input parameter to the program, so that the program can support different import countries.
- The new HTS effective dates will never chop up the effective dates of an existing HTS, and there will never be any rollback in dates. Therefore, a new HTS can only start in the middle of an existing HTS or cover a completely different time frame after the existing HTS.
- When loading a new HTS that starts in the middle of an existing HTS, the effect_to date of the existing HTS should be adjusted to one day before the new effect_from date.
- No existing HTS information should be purged by the program. It's the client's responsibility to handle that.

TABLE	SELECT	INSERT	UPDATE	DELETE
HTS	Yes	Yes	Yes	Yes
HTS_TAX	No	Yes	Yes	Yes
HTS_FEE	No	Yes	Yes	Yes
HTS_OGA	No	Yes	Yes	Yes
HTS_TARIFF_TREATMENT	Yes	Yes	Yes	Yes
HTS_TT_EXCLUSIONS	No	Yes	Yes	Yes
TARIFF_TREATMENT	Yes	No	No	No
COUNTRY_TARIFF_TREATMENT	Yes	No	No	No
HTS_CHAPTER	Yes	No	No	No
OGA	Yes	No	No	No

TABLE	SELECT	INSERT	UPDATE	DELETE
UOM_CLASS	Yes	No	No	No
CODE_DETAIL	Yes	No	No	No
QUOTA_CATEGORY	Yes	No	No	No
COUNTRY	Yes	No	No	No
HTS_CVD	No	No	Yes	No
HTS_AD	No	No	Yes	No
HTS_REFERENCE	No	No	Yes	No
ITEM_HTS	Yes	Yes	Yes	No
ITEM_HTS_ASSESS	No	No	Yes	No
ORDSKU_HTS	Yes	Yes	Yes	Yes
MOD_ORDER_ITEM_HTS	No	Yes	No	No
PERIOD	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No
DUAL	Yes	No	No	No
ORDSKU_HTS_ASSESS	No	No	No	Yes
ORDHEAD	Yes	No	No	No
ORDLOC	Yes	No	No	No
ORDSKU	Yes	No	No	No
CE_CHARGES	Yes	No	No	Yes
CE_ORD_ITEM	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Ad hoc

Scheduling Diagram: Run anytime as needed.

Pre-Processing: after hts upload conversion (hts2rms – PERL script).

Post-Processing: None

Threading Scheme: None

Restart Recovery

This program supports Retek standard intermittent commit and file upload restart/recovery. Recommended commit counter is 2000 (commit after every 2000 tariff records are read). Input file names must end in a “.1” for the restart mechanism to properly parse the file name. Since there is only 1 input file to be uploaded, only 1 thread is used. A reject file is used to hold records that have failed processing. The user can fix the rejected records and process the reject file again.

Program Flow

N/A

Shared Modules

ITEM HTS SQL.DELETE_ASSESS – given the item, hts, import_country_id, origin_country_id, effect_to and effect_from, this function deletes the corresponding record from item_hts_assess.

ITEM HTS SQL.DEFAULT_CALC_ASSESS – given the item, hts, import_country_id, origin_country_id, effect_to and effect_from, this function inserts into item_hts_assess, it also will potentially call other package functions and update other tables.

LC SQL.DELETE_LCORDAPP – given the order_no, this function deletes from lc_ordapply table.

OTB SQL.ORD_UNAPPROVE – given the order_no, this function updates the otb table.

ITEM_ATTRIB_SQL.GET_STANDARD_UOM – given the item_no, item_type and indicator, this function returns the standard_uom, standard_class, and conv_factor.

UOM SQL.CONVERT – given the to_uom, from_value, from_uom, item, supplier and origin_country, this function returns the to_value.

SQL_LIB.BATCH_MSG – returns error message information.

ORDER HTS SQL.DELETE_ASSESS -- given the order_no and seq_no, this function deletes from the ordsku_hts_assess table.

ORDER HTS SQL.DEFAULT_CALC_ASSESS -- given the order_no, seq_no, pack or item, hts, import_country_id, origin_country_id, effect_to and effect_from, this function inserts into ordsku_hts_assess, it also will potentially call other package functions and update other tables.

CE_CHARGES_SQL.INSERT_COMPS – given the ce_id, vessel_id, voyage_flt_ind, order_no, item, pack_item, hts, import_country_id, effect_from, effect_to, cvb_code this function inserts into the ce_charges table.

Function Level Description

Main

Standard Retek main function. This program takes in four parameters: userid/passwd, input file, reject file, import country id.

Init

A global variable is used to hold the import country id that is passed in as a program input parameter. Call check_country to make sure that import country exists on the COUNTRY table; return with fatal error if not. It is used as the import country throughout the program.

Open input file for read and open reject file for write.

Call retek_init() for restart/recovery initialization. If it is a fresh start, call retek_get_record to read the FHEAD line into the fhead structure.

Fetch vdate from period table. Fetch max_item from hts table and max ct from ordsku_hts and max ct from ce_charges. Fetch update_item_hts_ind and update_order_hts_ind from the system_options table. Call check_spi to make sure that 'C1' and 'C2' exist in the TARIFF_TREATMENT table as SPI's. 'C1' and 'C2' are default tariff treatments for every HTS. Return with fatal error if not.

File_process

Call function retek_get_record in a while loop to read the THEAD line into the thead structure: if the record type returned is 'FTAIL', exit the loop; set a save point. If the record type returned is 'THEAD', read the THEAD line into the thead structure that contains V1, V2, V3, V4 fields. The V4 record is not currently used in RMS/RTM. If the record type returned is other than 'FTAIL' or 'THEAD', give a fatal error (wrong record type).

Call function process_THEAD to further process data contained in the THEAD. Set process error flag to indicate non-fatal process error.

Call function retek_get_record in a while loop to read the TDETL line into the tdetl structure: if the record type returned is 'TTAIL', exit to the outer loop to continue reading THEAD records if any exists; if the record type returned is other than 'TDETL' or 'TTAIL', give a fatal error (wrong record type).

Call function process_TDETL to further process data contained in the TDETL.
Set process error flag to indicate non-fatal process error.

If update_item_hts_ind = "Y",

If tran_code is "A" or "R", call item_hts_update function. "A" stands for Update only and "R" stands for Replace. In both of these cases (as opposed to the other possibility of "D" for Delete) item tables will need to be updated.

If update_order_hts_ind = "Y", call ordsku_hts_search function.

If process error flag is set. Rollback database process to the save point. Write rejected records to the reject file.

Call restart_force_commit to perform intermittent commit for restart/recovery.

Process_THREAD

Fill the hts_keys structure with data from THREAD.

After fill in the hts_keys, verify that effect_from < effect_to date. If not, reject the record right away. Call valid_all_numeric function to check effect_from, and effect_to field. If invalid reject the record.

This function processes the information in V1, V2 and V3 records based on the transaction code ("A", "R", "D") in the V1 record. It compares the new effective dates against those of any existing HTS records with the same HTS code and import country.

If the transaction code type is 'A', insert a record into the HTS table; if the transaction code type is 'R', update the HTS record that has the same HTS code, import country id, effect_from and effect_to dates

For transaction code "A":

If new HTS covers a time period different than and after any existing HTS, or no HTS exists for the given HTS/import country, is a valid record for inserting.

If new the HTS record is overlapping with existing record and its effect_from date > existing record and effect_to >= existing effect_to date, it is a valid record. Process is as follows:

insert an HTS record with the same data as the existing overlapping HTS, except that the effect_to date should be 1 day before the effect_from date of the new HTS record;

Update the effect_to date of all corresponding child records to 1 day before the effect_from date of the new HTS record.

Insert new hts to the related tables.

Detailed technical description:

Call function `validate_hts_update` to verify that the record is valid for insert/update to the database or reject to the reject file. For the valid record call `hts_child_update` function to prepare child table processing.

Call `hts_table_insert` function to insert record to the hts table. if any invalid information exists, write to error file.

Call `hts_oga_insert` function to insert record/s to the hts_oga table. if any invalid information exists, write to error file.

Call `hts_spi_insert` function to insert record/s to the hts_tariff_treatment table. if any invalid information exists, write to error file.

Call `hts_gsp_insert` function to insert record/s to the hts_tt_exclusions table. if any invalid information exists, write to error message log file.

Set process error flag if non fatal error occurs. Return error flag.

For transaction code "R":

Search for the HTS with the same HTS, import country id, effect_from and effect_to dates. If no record found, reject the record.

If a record is found, delete the following child table records with the same HTS, import country id, effect_from and effect_to dates:

insert to update the HTS table and re-insert child table information from the input file.

Detailed Technical Description:

Call function `search_hts_update` to find record that can be updated in the database tables.

If one exists, prepare child tables for processing.

Call `hts_table_insert` function to insert record to the hts table. if any invalid information exists, write to error file.

Call `hts_oga_insert` function to insert record/s to the hts_oga table. if any invalid information exists, write to error file.

Call `hts_spi_insert` function to insert record/s to the hts_tariff_treatment table. if any invalid information exists, write to error file.

Call `hts_gsp_insert` function to insert record/s to the hts_tt_exclusions table. if any invalid information exist, write to error message log file.

Set process error flag if non fatal error occurs. Return error flag.

For transaction code "D":

Search for the HTS with same HTS, import country id , effect_from and effect_to dates.

If a record is found update HTS and all its child records to yesterday.

Detailed Technical Description:

Call function search_hts_reset to find updateable record in the hts table. If one exists, insert new hts record. Call function hts_child_update to update all the child records, then delete the existing hts record.

Validate_hts_update

new HTS starts before or on the same day as any existing HTS, or

new HTS starts after and ends before any existing HTS:

effect_from >= new effect_from OR

effect_from < new effect_from and effect_to > new effect_to

This is an invalid record. Write the record to the reject file, write an error message to the message log file, and return to the calling function with a non-fatal error.

new HTS starts after and overlaps with an existing HTS:

effect_from < new effect_from and effect_to >= new effect_from or new HTS starts after old end date and therefore does not overlap at all. The ranges are completely separate.

This is a valid record, and a most likely scenario. Fetch the effect_from and effect_to of the existing HTS. Insert a new record with effect_from date same as existing overlapping hts record and effect_to date is 1 day before the new effect_from date to hts table.

Call function hts_child_update function to update effect_to date of all child records to 1 day before the new effect_from date.

Delete the old record from hts table.

Search_hts_update

Search for the HTS with the same HTS, import country id, effect_from and effect_to dates. If no record found, reject the record.

If a record is found, delete the following child table records with the same HTS, import country id, effect_from and effect_to dates:

- HTS_TT_EXCLUSIONS
- HTS_TARIFF_TREATMENT
- HTS_OGA
- HTS_TAX
- HTS_FEE

Note: HTS table record cannot be deleted due to the other child tables on HTS: ITEM HTS, ITEM HTS_ASSESS, ORDSKU HTS, HTS_CVD, HTS_AD, HTS_REFERENCE, HTS_CHAPTER. The information on these tables won't be loaded in the HTS upload process.

Seach_hts_reset

Search for the HTS with the same HTS, import country id, effect_from and effect_to dates. If no record found, reject the record.

Insert into HTS, all the same information, but inserting yesterday as the new to_date.

If a record is found, call hts_child_update function to update the records in the child tables with effect_to date to yesterday:

Hts_child_update

This function updates the effect_to date of the existing overlapping HTS record on child tables. Since the child tables have referential constraints on the effective dates of the parent table HTS.

Update the effect_to date of all corresponding child records to 1 day before the effect_from date of the new HTS record.

The following child tables should be updated:

- HTS_TARIFF_TREATMENT
- HTS_TT_EXCLUSIONS
- HTS_AD
- HTS_CVD
- HTS_OGA

- HTS_REFERENCE
- HTS_TAX
- HTS_FEE
- ITEM HTS
- ITEM HTS_ASSESS
- ORDSKU HTS
- CE_CHARGES

Note: Since table HTS_TT_EXCLUSIONS has a foreign key on the effect_to date of table HTS_TARIFF_TREATMENT, we cannot update the effect_to date of HTS_TARIFF_TREATMENT directly. Likewise, insert an HTS_TARIFF_TREATMENT record with the new effect_to date first; then update the effect_to date of the HTS_TT_EXCLUSIONS table; at the end delete the HTS_TARIFF_TREATMENT record with the original effect_to date.

Call delete_ord_temp_tables and pass in the value “-1” because there is no known order_no at this point.

Item_hts_update

Call size_item_array function to allocate space for the items

Fetch item, origin_country_id and status from item_hts into struct

If no data found, call free_itemlist and go to the next record. If data is found,

Loop

If tran_code = “A” the item will need to be inserted with the same data as the fetched record but with new effect_to and effect_from dates.

Insert dates into item_hts

Delete old record from item_hts

call the package SQL Delete assess to delete the old records from item_hts_assess.

If tran_code = “R”

Call SQL Delete_assess to delete the old records from item_hts_assess

Call SQL Default_calc_assess to update the item_hts_assess table (ie insert record with new dates and recalculate)

Call ECL_CALC_SQL.CALC_COMP to recalculate expenses based on new assesses.

Insert into mod_order_item_hts a new record with same data but new dates.

Call free_itemlist

Ordsku_hts_search

Call size_ord_array function to allocate space for the order information

Fetch values from ordhead, ordsku_hts, ordsku and ordloc into struct (all necessary values to be able to do a complete insert into the mod_order_item, ordsku_hts, and ordsku_hts_assess tables.

If no data found, call free_ordlist and go to next record. If data is found,

Loop

If order status = "A", (the order needs to be updated) set status from approved back to worksheet by calling SQL functions (LC_SQL.DELETE_LCORDAPP and OTB_SQL.ORD_UNAPPROVE).

Insert into mod_order_item_hts table (just the order_no and indicator set to 'Y')

Call ordsku_hts_update

Call free_ordlist

Ordsku_hts_update

Call size_ce_array to allocate space for the custom entry information

Fetch custom entry values from ce_ord_item, ce_head, item_supp_country into struct

If no data found, call ordhts_update. If data is found,

If CE status = "W", (worksheet status)

Call ordhts_update

Loop for each custom entry record

Call ce_update

If status != "W" then the quantity cleared will need to be compared to the total quantity. In order to do that they will need to be converted to the standard uom format

Loop

Call uom_convert to get the total quantity.

If total_qty < qty_ordered

Call ordhts_update

Call free_ceordlist

Ordhts_update

if tran_code = "A" or "D"

Delete old record (record with old dates) from ordsku_hts_assess

Delete old record (record with old dates) from ordsku_hts

Insert record with new dates into ordsku_hts

Else if tran_code = "A"

Insert record with new dates into ordsku_hts

else if tran_code = "D"

Call SQL Delete_assess by calling order_del_assess function

Call SQL calc_comp

If the item is a pack item check to see if a record already exists on mod_order_item_hts – if it does not, insert one with the pack_item

If it is not a pack item, insert with item_no into mod_order_item_hts.

Return 0

Else if tran_code = "R"

Call delete_ord_temp_tables and pass in the order_no.

Call SQL Delete_assess by calling order_del_assess function

Call ORDER HTS SQL.DEFAULT_CALC_ASSESS with either the pack_no or item_no depending on if it is a pack or not.

Call ELC_CALC.CALC_COMP

If it is a pack item insert into mod_order_hts with the pack_no

If it is not a pack item, insert into mod_order_item_hts with the item_no

Ce_update

1. Delete from ce_charges.
2. If it is a “D”, call CE_CHARGES_SQL.INSERT_COMPS

Hts_table_insert

Before inserting into or updating the HTS table,

Call function check_chapter to make sure that the chapter already exists on the HTS_CHAPTER table. If not, reject the record;

Call check valid_all_numeric function to check unit for all numeric value.

Call function check_uom to make sure that the UOMs (UOM1, UOM2, UOM3) already exist on the UOM_CLASS table. Reject the record if UOM does not exist.

Call function check_duty to make sure that the duty code already exists on the CODE_DETAIL table. If not, reject the record.

Call valid_all_numeric function to verify that the quota is all numeric. Then calling function check_quota to make sure that the quota category already exists on the QUOTA_CATEGORY table. If not, reject the record.

Update the existing hts record with the updated hts_desc, chapter, units, units_1, units_2, units_3, duty_comp_code, more_hts_ind, quota_cat, quota_ind, ad_ind, cvd_ind.

Insert the following into the HTS table:

hts: tariff number (V1c)

import_country_id: import country from the program input parameter

effect_from: begin effective date (V1e)

effect_to: end effective date (V1f)

hts_desc: commodity description (V1l)

chapter: 1st 4 (leftmost) digits of tariff number

units: number of reporting units (V1g)

units_1: first unit of measure (V1h) (If the number of reporting units is zero, this should be defaulted to ‘X’)

units_2: second unit of measure (V1i)—NULL if not given

units_3: third unit of measure (V1j)—NULL if not given

duty_comp_code: duty code (V1k)

more_hts: Y if additional tariff indicator (V2j) is 'R', N otherwise

quota_cat: category number (V3h) but only if quota indicator (V3g) is 1

quota_ind 'Y' if there is a quota, 'N' otherwise

ad_ind 'Y' if the anti-dumping flag (V3f) is 1, N otherwise

cvd_ind 'Y' if the countervailing duty flag (V2k) is 1, N otherwise

Hts_oga_insert

For each OGA code, call function check_oga to verify that the OGA code exists on the OGA table. If not, reject the record; otherwise, call hts_oga_insert to insert into HTS_OGA.

Insert the following into the HTS_OGA table:

hts: tariff number (V1c)

import_country_id: import country from the program input parameter

effect_from: begin effective date (V1e)

effect_to: end effective date (V1f)

code: OGA code from OGA codes field (V3f)

reference_id: NULL

comments: NULL

Hts_spi_insert

For each SPI, call function check_spi to check if the SPI exists on the tariff_treatment table; if not, reject the record. Call function hts_tariff_treatment_insert to insert into HTS_TARIFF_TREATMENT. In addition to the SPI records in V3, 'C1' and 'C2' are default tariff_treatments for every HTS. So, two extra records should be inserted into HTS_TARIFF_TREATMENT with SPI codes 'C1' and 'C2'. 'C1' takes the special_duty_rate from V1 and Column 1 rates from V2; 'C2' takes Column 2 rates from V2.

Before inserting, call function check_spi to make sure that the SPI code (tariff treatment) exists on the TARIFF_TREATMENT table; reject the record if it does not.

Call valid_all_numeric function to check specific_rate, ad_rate, other_rate for all numeric value. If not, reject the record.

Reject HTS lines that have rate greater than 9999999999. A brief explanation of why this is done is located at the end of the function level description section.

Insert the following into the HTS_TARIFF_TREATMENT table:

hts: tariff number (V1c)

import_country_id: import country from the program input parameter

effect_from: begin effective date (V1e)

effect_to: end effective date (V1f)

tariff_treatment: SPI code from V3i

specific_rate: 0,col1 or col2 specific rate, as appropriate (0 for SPI's,col 1 for col1, col 2 for col2)

av_rate: 0,col1, or col2 ad valorem rate, as appropriate (0 for SPI's)

other_rate: 0,col1, or col2 other rate, as appropriate (0 for SPI's)

Hts_gsp_insert

For each GSP excluded country, call function check_country_tariff_treatment to check that the country and tariff treatment combination exists on the COUNTRY_TARIFF_TREATMENT table; if not, reject the record.

Insert the following into the HTS_TT_EXCLUSIONS table

hts: tariff number (V1c)

import_country_id: import country from the program input parameter

effect_from: begin effective date (V1e)

effect_to: end effective date (V1f)

tariff_treatment: first SPI code from V3i

origin_country_id: excluded country code from V3d (GSP excluded countries)

Check_spi

Check to see if SPI exists on TARIFF_TREATMENT table; reject the record if it doesn't.

Check_country

Check to see if country exists on COUNTRY table; reject the record if it doesn't.

Check_chapter

Check to see if chapter exists on the HTS_CHAPTER table and reject the record if it doesn't.

Check_uom

Check to see if uom exists on UOM_CLASS table; reject the record if it doesn't.

Check_duty

Check to see if duty code exists on CODE_DETAIL table (check for the code where code_type='DCMP'); reject the record if it doesn't.

Check_quota

Check to see if the quota_category exists on the QUOTA_CATEGORY table; reject the record if it doesn't.

Check_oga

Check to see if the oga code exists on the OGA table; reject the record if it doesn't.

Check_comb_country_tt

Check to see if the country and tariff_treatment combination exists on the COUNTRY_TARIFF_TREATMENT table; reject the record if it doesn't.

Process_TDEL

Format the tax line information from tdel structure.

Call function process_taxfee, if no non-fatal error in the process_THREAD function.

Process_tax_fee

If tax specific rate or tax ad rate is not null, call hts_taxfee_insert to insert the tax rates into HTS_TAX or HTS_FEE tables. If special rates exist on the tax line, call function hts_tariff_treatment_insert to insert into the HTS_TARIFF_TREATMENT table using the ISO country code as the tariff treatment (SPI). If the SPI given on the tax line already exists for the HTS, the record should be updated, as the tax line special rate takes precedence over the V3 line SPI's rate

Call valid_all_numeric function to check tax_specific_rate, tax_av_rate, fee_specific_rate, fee_av_rate for all numeric value, if not reject the record.

Reject HTS lines that have rate greater than 9999999999. A brief explanation of why this is done is located at the end of the function level description section.

Hts_taxfee_insert

If the tax class code is 016,017,018,or 022 it is a tax; insert into HTS_TAX

If the tax class code is 038,053,054,055,056,057,079,090,103 it is a fee; insert into HTS_FEE

Insert the following into the HTS_TAX or HTS_FEE table:

hts: tariff number (V1c)

import_country_id: import country from the program input parameter

effect_from: begin effective date (V1e)

effect_to: end effective date (V1f)

tax_type/fee_type: tax class code (V5h)

tax_comp_code/fee_comp_code: tax comp code (V5i)

tax_specific_rate/fee_specific_rate: tax specific rate (V5k)

tax_av_rate/fee_av_rate: tax ad valorem rate (V5l)

Hts_tariff_treatment_insert

Before calling this function, call function check_spi to make sure that the SPI code (tariff treatment) exists on the TARIFF_TREATMENT table; reject the record if it does not.

Insert the following into the HTS_TARIFF_TREATMENT table:

hts: tariff number (V1c)

import_country_id: import country from the program input parameter

effect_from: begin effective date (V1e)

effect_to: end effective date (V1f)

tariff_treatment: SPI code from V3i

specific_rate: 0,col1 or col2 specific rate, as appropriate (0 for SPI's,col 1 for col1, col 2 for col2)

av_rate: 0,col1 or col2 ad valorem rate, as appropriate (0 for SPI's)

other_rate: 0,col1 or col2 other rate, as appropriate (0 for SPI's)

Size_item_array

Allocates space for the item array struct

Size_ord_array

Allocates space for the order array struct

Size_ce_array

Allocates space for the custom entry array struct

Free_orditemlist

Frees the space in the array

Free_itemlist

Frees the space in the array

Free_ceordlist

Frees the space in the array

Uom_convert

Calls ITEM_ATTRIB_SQL.GET_STANDARD_UOM

Calls UOM_SQL.CONVERT

Order_del_assess

Calls ORDER HTS_SQL.DELETE_ASSESS

Delete_ord_temp_tables

If an order no is not passed in, look at the hts table and see if there is an order that exists for that hts. If so, loop and for each record see if there is a record to delete on the temp tables by calling
ORDER_SETUP_SQL.DELTE_TEMP_TABLES.

If the order number was passed in, call
ORDER_SETU_SQL.DELETE_TEMP_TABLES right away.

Final

Restart/recovery close and close input and reject file.

Why HTS lines that have a rate greater than 9999999999 need to be rejected

For fields `specific_rate`, `av_rate`, `other_rate`, RMS has the data type `Number(12,8)` and numbers coming in from the customs tape also have 8 implied digits. However, when storing the number into the Retek database, we need to divide the number coming in from the customs tape by 1000000 (left shift 6 digits) instead of 100000000 (left shift 8 digits). This is because Retek stores the percent part of the rate only. In other words, rate 11.5% (0.115) is stored as 11.5 in Retek database, whereas it will come in from the customs tape as 11500000 (=0.115). Therefore, the highest rate that can be represented in Retek is 9999.99999999% (= 99.999999999, or < 100 times). So we need to reject HTS lines that have rate greater than 9999999999.

Note: This is true for `hts spi` and `hts tax/fee specific_rate` and `av_rate`, except that when 999999999999 (12 nines) are used, it represents a special code for NULL.

I/O Specification

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record descriptor	Char(5)	FHEAD	Describes file line type
	Line number	Number(10)	0000000001	Sequential file line number
	Retek file ID	Char(5)	HTSUP	Describes file type
THEAD	Record descriptor	Char(5)	THEAD	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction id	Number(10)		Unique transaction id
	HTS Line	Char(320)		V1 through V4 records from the customs HTS file concatenated together
TDETL	Record descriptor	Char(5)	TDETL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction id	Number(10)		Unique transaction id
	Tax/fee line	Char(80)		V5 through V9 records from the customs HTS file, each on a separate TDETL line
TTAIL	Record descriptor	Char(5)	TTAIL	Describes file line type

Record Name	Field Name	Field Type	Default Value	Description
	Line number	Number(10)		Sequential file line number
	Detail lines	Number(6)		Number of lines between THEAD and TTAIL
FTAIL	Record descriptor	Char(5)	FTAIL	Describes file line type
	Line number	Number(10)		Sequential file line number
	Transaction lines	Number(10)		Number of lines between FHEAD and FTAIL

Note: The input file contains lines of 720 characters, i.e. the newline character occurs only after every 720 characters. Each 720-character line consists of nine 80-character records. Each 80-character record starts with ‘V1’ or ‘V2’ ... or ‘V9’ or blank if the record is completely empty. For each tariff, records V1 and V2 are mandatory; records V3 through V9 are optional, which means they can be all blank. Record V4 is not currently used in RMS/RTM. Records V5 through V9 contain the tax/fee information for the tariff, and all have the same structure. The lower-case letters in the record name block are as a convenience to cross-reference with the US Customs file description.

Record Name	Field Name	Field Type	Default Value	Description
V1 a	Control identifier	Char(1)	V	Identifies start of record
b	Record type	Char(1)	1	Identifies record type
c	Tariff number	Number(10)		unique tariff number—will be left justified and space filled if less than 10 digits
d	transaction code	Char(1)	A, D, R	Valid codes are: A = Add D = Delete R = Replace
e	begin effective date	char(6)		MMDDYY format—date this tariff record becomes effective
f	end effective date	char(6)		MMDDYY format—last date this tariff record becomes effective; can be ‘999999’
g	number of reporting units	number(1)	0,1,or 2 or 3	Number of census reporting units required
h	1 st reporting unit of measure	char(3)		unit of measure (“X” if none required)
I	2 nd reporting unit of measure	char(3)		unit of measure—may be blank
j	3 rd reporting unit of measure	char(3)		unit of measure—may be blank
k	duty computation code	char(1)		Indicates formula to compute duty
l	commodity description	char(30)		Description of tariff commodity

Record Name	Field Name	Field Type	Default Value	Description
m	column 1 specific rate of duty	Number(12)		specific rate listed in general column 1 rate of duty 8 implied decimal places
n	base rate indicator	char(1)	'B' or blank	If B, duty rate involves a base rate. Not used in RMS.
o	space fill	char(1)	blank	Not used in RMS.
V2 a	Control identifier	char(1)	V	Identifies start of record
b	Record type	char(1)	2	Identifies record type
c	tariff number	Number(10)		Unique tariff number
d	general column 1 ad valorem percentage	Number(12)		ad valorem rate 8 implied decimal places
e	column 1 other	Number(12)		specific rate if exists, or other ad valorem percentage if needed for duty computation 8 implied decimal places
f	Column 2 specific rate	Number(12)		specific rate 8 implied decimal places
g	Column 2 ad valorem percentage	Number(12)		ad valorem percentage 8 implied decimal places
h	Column 2 other rate	Number(12)		other specific rate or ad valorem percentage 8 implied decimal places
i	countervailing duty flag	char(1)	blank or 1	subject to CVD if 1; space fill otherwise
j	additional tariff indicator	char(1)	blank or 'R'	space fill if no additional tariff number needed; 'R' if it is
k	space fill	char(6)	blanks	Not used in RMS.
V3 a	Control identifier	char(1)	V	identifies start of record
b	Record type	char(1)	3	identifies record type
c	tariff number	Number(10)		unique tariff number

Record Name	Field Name	Field Type	Default Value	Description
d	GSP excluded countries	char(20)		Up to 10 2-char ISO country codes
e	OGA codes	char(15)		Up to 5 other gov agency codes
f	anti-dumping flag	char(1)	1 or blank	if 1, may be subject to dumping duties
g	quota indicator	char(1)	1 or blank	if 1, tariff subject to possible quota
h	category number	Number(3)		quota category
I	special program indicators	char(20)		up to 10 2-char codes
j	space fill	char(8)	blanks	Not used in RMS.
NEWLINE			\n	
V4 a	Control identifier	char(1)	V	identifies start of record Entire V4 record not used in RMS.
b	Record type	char(1)	4	identifies record type
c	tariff number	number(10)		unique tariff number
D	value edit code	char(3)		defines value edit
E	value low bounds	number(10)		minimum or lower value to use in edit (can be blank) 5 implied decimal places
F	value high bounds	number(10)		max or higher value to use in edit (can be blank) 5 implied decimal places
G	entry date restriction	number(1)	0,1, or 2	date edit code
H	beginning restriction date	char(4)		MMDD format—beginning/earliest date for edit (can be blank)
I	end restriction date	char(4)		MMDD format—end/latest date for edit (can be blank)
J	entry date restriction 2	number(1)	0,1, or 2	date edit code (can be blank)
K	beginning restriction date 2	char(4)		MMDD format—beginning/earliest date for edit (can be blank)

Record Name	Field Name	Field Type	Default Value	Description
L	end restriction date 2	char(4)		MMDD format—end/latest date for edit (can be blank)
M	country of origin	char(2)		country of origin ISO code or 01 (must be Col 1 country) or 02 (must be Col 2 country)
N	space filler	char(2)	blanks	
o	quantity edit code	char(3)		defines quantity edit
p	low quantity	number(10)		min/lower qty for edit 5 implied decimal places
q	high quantity	number(10)		max/higher qty for edit 5 implied decimal places
V5 a	Control identifier	char(1)	V	identifies start of record
b	Record type	char(1)	5,6,7,8,9	identifies record type
c	tariff number	number(10)		unique tariff number
d	Country code	char(2)		ISO code of country or ‘E ‘ (E plus space) or ‘J ‘ (J plus space) where special rate applies
e	specific rate	number(12)		Col 1 special rate 8 implied decimal places
f	ad valorem rate	number(12)		Col 1 special ad valorem rate 8 implied decimal places
g	Other rate	number(12)		Col 1 other special rate if needed 8 implied decimal places
h	tax/fee class code	char(3)		Identifies specific tax or user fee
I	tax/fee comp code	char(1)		used to compute amt of tax/fee
j	tax/fee flag	number(1)		1—required 2—may be required Not used in RMS.
k	tax/fee specific rate	number(12)	blank if no value	Specific rate to calculate tax or fee 8 implied decimal places

Record Name	Field Name	Field Type	Default Value	Description
l	tax/fee ad valorem	number(12)	blank if no value	ad valorem rate to compute tax or fee 8 implied decimal places
m	space fill	char(1)	blank	
***** V6 through V9 records have the same fields as the V5 record. *****				

Inventory Adjustment Purge [invaprg]

Design Overview

The Inventory Adjustment Purge (invaprg) module will delete all obsolete inventory adjustment records where a pre-determined number of months have elapsed. The pre-determined number of months is a system option.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
INV_ADJ	No	No	No	No	Yes
PERIOD	No	Yes	No	No	No
RESTART_CONTROL	No	Yes	No	No	No
UNIT_OPTIONS	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE AD-HOC (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

In the init() function, select the number of months inventory adjustments are held is selected from the UNIT_OPTIONS table.

Then, in the process() function, all records are delete from the INV_ADJ table where the number of months between the current date and the adjustment date is greater than the number of months that inventory adjustments are held.

I/O Specification

N/A

Technical Issues

N/A

Stock Adjustment Update [invaupld]

The Inventory Adjustment Upload (invaupld) module processes inventory adjustments initiated in an external system (i.e. WMS or POS) and makes the required adjustments in the RMS. There are two types of adjustments that can be made – adjustments to total stock on hand or adjustments to unavailable inventory. Adjustments to total stock on hand remove the inventory for the specified item/location from the system. Adjustments made to unavailable inventory do not remove stock on hand for the specified item, but do adjust the unavailable inventory for the item/location. Transaction-level stock ledger records are also written for each adjustment.

Design Overview

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
RAG_SKUS_ST	No	Yes	No	Yes	No
WIN_STORE	No	Yes	No	Yes	No
WIN_WH	No	Yes	No	Yes	No
TRAN_DATA	No	No	Yes	No	No
RAG_SKU	No	Yes	No	No	No
RAG_STYLE_ST	No	Yes	No	No	No
RAG_STYLE_WH	No	Yes	No	No	No
INV_ADJ	No	No	Yes	No	No
INV_STATUS_QTY	No	No	Yes	Yes	No

Scheduling Constraints

Processing Cycle: Daily

Scheduling Diagram: This program will be run at the beginning of the batch run during the POS polling cycle, or possibly at the end of the batch run if there are pending warehouse transactions. It can also be scheduled to run multiple times throughout the day, as WMS or POS data becomes available.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

The logical unit of work for the inventory adjustment module is the discrete inventory adjustment transaction. A unique transaction set number generated by the external system will identify each adjustment. This transaction will be defined as the logical unit of work. If any portion of the processing for the complete transaction fails, the entire inventory adjustment must be re-processed.

A save-point will be issued prior to processing a new transaction. If any record within the transaction fails, the whole transaction will be rolled back to the most recent save-point. This way, the successfully processed transactions will remain posted to the database but not yet committed.

To prevent excessive rollback space usage, intermittent commits will be issued based on a commit counter. The recommended commit counter setting is 10000 records (subject to change based on experimentation). The commit counter is based on actual records processed, not overall transactions, nor the number of writes to the database, since the database interactions will be a constant multiplier of the commit counter. A transaction cannot be committed to the database until it is complete so the commit counter is viewed as a minimum threshold, that once reached, will force a commit after the completion of the current transaction.

Error handling will be based on the logical unit of work as well. If a given record within a transaction fails, that error will be posted to the standard error log for the batch module. If the error is of a non-fatal type, all subsequent detail records within the transaction will continue to be processed and any errors noted will continue to be posted. After processing all errors for the transaction, the entire transaction will be rejected to a runtime specified rejection file. If a fatal error is encountered, the file pointer at the time of the last commit will have been posted to the bookmark and all transactions from the last commit will be rolled back. Processing will commence with from the saved file position.

Shared Modules

ITEMLOC_QUANTITY_SQL.GET_STOCK_ON_HAND

ITEMLOC_QUANTITY_SQL.GET_UNAVAILABLE

INVADJ_VALIDATE_SQL.REASON_EXIST

INVADJ_VALIDATE_SQL.SKU_LOC_EXIST

ITEM_ATTRIB_SQL.UPC_GET_SKU

ITEM_ATTRIB_SQL.TYPE

INVADJ_SQL.ADJ_STOCK_ON_HAND

UPDATE_SNAPSHOT_SQL.EXECUTE

INVADJ_SQL.ADJ_UNAVAILABLE

INVADJ_SQL.ADJ_TRAN_DATA

Function Level Description

init()

This function initializes restart recovery, gets system information and opens I/O files.

process_file()

The input file will be read in blocks in this function. The size of the blocks is to be determined by the commit counter value on the RESTART_CONTROL table (this is initialized in the init function). Once the block has been read from the file into an array of structures, calling the process_record function will process each record structure. The external functions restart_commit and process_dtl_ftail are called to perform intermittent commits and validate input file integrity respectively.

process_record()

Each record is initialized and validated. (The program units init_variables() and validate_fields are called to perform this function.) If the inventory status is normally available stock (inv_status variable passed in the input file is 0), the ITEM_LOC_QTY_SQL.GET_STOCK_ON_HAND package is called to get the stock on hand, otherwise, the stock amount is retrieved from the inv_status_qty table. The sku information is inserted into the inv_adj table. Finally if the stock is available, the process_available function is called, otherwise the process_unavailable function is called.

Validate_fields

The following external functions are called to validate input fields:

INVADJ_VALIDATE_SQL.REASON_EXIST

- validates the reason for the stock adjustment

ITEM_ATTRIBUTE_SQL.UPC_GET_SKU

- gets Retek SKU value from UPC value

ITEM_ATTRIBUTE_SQL.TYPE

-retrieves the merchandise type of the SKU

INVADJ_VALIDATE_SQL.SKU_LOC_EXIST

-validates that the SKU exists at the location

process_available()

The following external functions are called to process the available stock:

INVADJ_SQL.ADJ_STOCK_ON_HAND

- adjusts the available stock on hand at the location

INVADJ_SQL.ADJ_TRAN_DATA

- writes tran_data record to adjust financials

UPDATE_SNAPSHOT_SQL.EXECUTE

- updates edi_daily_sales & stake_sku_loc snapshots

process_available()

The following external functions are called to process the available stock:

INVADL_SQL.ADJ_UNAVAILABLE

- adjusts the unavailable stock on inv_adj_qty

INVADJ_SQL.ADJ_TRAN_DATA

- writes tran_data record to adjust financials

ON Fatal Error:

Rollback to last physical commit-point

Exit Program

ON Non-Fatal Error:

Rollback to last save-point

Write out complete transfer transaction to the reject file, pass file pointer at last save-point and current file pointer

I/O Specification

Input File

The input file should be accepted as a runtime parameter at the command line.

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	INVA	Identified file as 'Inventory Adjustments'
	File Create Date	Date	create date	date file was written by external system
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Number(14)	specified by external system	used to force unique transaction check
	Transaction Date	Date	specified by external system	date of the inventory adjustment
	Location Type	Char(2)	ST - store WH - warehouse	specifies the type of location
	Location Value	Number(4)	location identifier	specifies the location id number
	Item Type	Char(3)	UPC SKU	item type will be represented as an UPC or an SKU
	Item Value	Number(13)	item identifier	the id number of a SKU or UPC
	UPC Supplement	Number(5)	upc supplement identifier	used to further specify the id of an UPC item

Record Name	Field Name	Field Type	Default Value	Description
	Inventory Status	Number(2)	status code type	inventory status code as specified in the inventory status code table - 0 for total stock on hand adjustments
	Adjustment Reason	Number(2)	reason code	adjustment reason code as specified in the adjustment reason code table (reason 98 is for internal use only)
	Adjustment sign	Character(1)	+, -	sign of adjustment qty
	Adjustment Quantity	Signed Number(12,4)	inventory adjustment value	number of units to be adjusted of the given item at the given location
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Line Count	Number(10)	total detail + transaction head lines	sum of all detail lines, excluding file header and trailer

Reject File

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will need to be created by the transfer out module and a reject line counter will be required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and trailer record and the file line count will be equal to 0. The reject filename should also be specified as a runtime parameter.

Error File

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

N/A

Close Unmatched Shipments [invclshp]

Design Overview

This batch program will close all shipments that have remained open for a specified number of days and are not associated with any open invoices. This will be accomplished by setting the `invc_match_status` on the `SHIPMENT` table to 'C'losed.

Scheduling Constraints

Processing Cycle:	ad hoc
Scheduling Diagram:	after invpost
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

For all shipments with an `invc_match_status` != 'C' on SHIPMENT, set the shipment's `invc_match_status` to 'C' if all of the following conditions is met:

- The shipment is in 'R' (received) status.
- The shipment's `receive_date` on SHIPMENT is `system_options.close_open_ship_days` before today's date.
- No invoice matched to the shipment is not in 'P' status.
- The shipment is not associated with any open (unposted) invoices on the `INVC_XREF` table.
- All shipments that are in 'I' nvoice status but are associated with closed orders will also be purged from the system.

Technical Issues

N/A

Invoice Post [invcpost]

Design Overview

This batch program will insert records into the IIF (invoice interface to financials) staging tables. It will insert new invoices with a payment status of 'H' (hold payment) so that the AP system can take all current/future liabilities into consideration. It will also insert approved matched invoices or force-paid invoices with a payment status of 'P' (ready to pay). Invoices from suppliers that are marked as pre-paid suppliers will be inserted with a payment status of 'U' (pre-paid unmatched) if unapproved or 'M' (pre-paid matched) if approved.

Approved invoices associated with approved debit/credit memos or credit notes will be inserted with a payment status of 'P' (ready to pay) since issuing a debit/credit memo should release the invoice for payment. However, the invoices and the debit/credit memos or credit notes associated should all be in approved status to be posted in tandem. Invoices with attached credit note requests will only be posted when an approved merchandise credit note has been attached. Credit note requests are never sent to financials. All types of invoices (except credit note requests), including debit/credit memos or credit notes, will be sent to the IIF staging tables even if they are not qualified for being posted.

Modification(by Hien Tran Feb-07-2000)

This program will be modified to also post INVC_DISCOUNT records to the IIF_DISCOUNT table when posting an invoice.Scheduling Constraints.

Processing Cycle: daily

Scheduling Diagram: after ediupinv and invmtch, before interface to financial system

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: None

Restart Recovery

The logical unit of work for the invoice post module will be the invoice transaction. Restart/recovery will be based on the invoice transaction. In order to bundle an invoice and its associates, the invoice transaction here means the invc_id itself for an invoice not referencing another invoice and the ref_invc_id for an invoice that references another invoice. Therefore, one logical unit of work can contain multiple invoices.

The driving cursor for this restart/recovery strategy looks like the following. Note that the second decode() of the ORDER BY clause is used to put an invoice ahead of its associates if it is referenced.

```
SELECT      ih.inv_c_id,
            ih.inv_c_type,
            ih.supplier,
            ih.partner_type,
            ih.partner_id,
            ih.ext_ref_no,
            ih.ref_inv_c_id,
            ih.ref_rtv_order_no,
            ih.ref_price_change,
            ih.ref_rsn_code,
            ih.terms,
            TO_CHAR(ih.due_date, 'YYYYMMDD'),
            ih.payment_method,
            ih.terms_dscnt_pct,
            ih.terms_dscnt_appl_ind,
            ih.freight_terms,
            TO_CHAR(ih.inv_c_date, 'YYYYMMDD'),
            ih.force_pay_ind,
            ih.post_date,
            ih.currency_code,
            ih.exchange_rate,
            ih.total_merch_cost,
            ih.total_qty,
            ih.comments,
```



```

        ih.status,
        ih.direct_ind,
        ih.paid_ind,
        ih.addr_key,
        ih.payment_ref_no,
        ih.payment_date,
        ih.proof_of_delivery_no
FROM      invc_head ih
WHERE     ih.invc_type != 'R'
AND       (ih.status = 'A'
OR  (ih.force_pay_ind = 'Y'
AND ih.status != 'P')
OR  ih.post_date is NULL)
AND       decode(ih.ref_invc_id,
                NULL, ih.invc_id,
                ih.ref_invc_id) > NVL(:ps_restart_invc_id, -999
ORDER BY decode(ih.ref_invc_id, NULL, ih.invc_id, ih.ref_invc_id),
        decode(ih.ref_invc_id, NULL, 0, 1);

```

The commit_max_ctr field on the restart_control table will determine the number of transactions that equal a logical unit of work. It should be set to prevent excessive rollback space usage. The recommended commit counter setting is 10000 records (subject to change based on experimentation).

Program Flow

N/A

Shared Modules

N/A

Function Level Description

init():

Retrieve system date and VAT indicator from PERIOD and SYSTEM_OPTIONS:

vdate

vat_ind

process():

In a LOOP, fetch the driving cursor. For each invoice fetched, insert into arrays (for insert into the IIF tables) as follows:

For the insert into IIF_HEAD, gather the following values for each invoice:

invc_id = invc_head.invc_id

invc_type = invc_head.invc_type

supplier = invc_head.supplier

partner_type = invc_head.partner_type

partner_id = invc_head.partner_id

ext_ref_no = invc_head.ext_ref_no

payment_status will be decided by the action taken on this invoice, invc_head.direct_ind, and the prepay indicator:

Action is POST if

The invoice is being force-paid (force_pay_ind = 'Y')

The invoice is in 'A' status and not associated with any other invoices

The invoice is in 'A' status, associated with other invoices and all its associates are ready (debit memos/credit memos/credit notes are all in 'A' status, and, if any credit note request exists, debit memo or credit note in 'A' status exist).

The invoice is in 'A' status and referencing an invoice in 'P' status (either just posted or previously posted).

Action is SEND if the invoice is not qualified for being POSTed but has not been sent yet

Action is NONE if the invoice is not qualified for being POSTed and has been sent already

The prepay indicator indicates if a supplier is marked for pre-payment. It is fetched from `supsup.prepay_inv_ind` for a supplier invoice; it is always 'N' for a non-supplier invoice

`Payment_status` is 'C' if `inv_head.paid_ind` is 'Y' for the invoice

`Payment_status` is 'M' if the action is POST and prepay indicator is 'Y'

`Payment_status` is 'P' if the action is POST and prepay indicator is 'N'

`Payment_status` is 'U' if the action is SEND and prepay indicator is 'Y'

`Payment_status` is 'H' if the action is SEND and prepay indicator is 'N'

`ref_inv_id` = `inv_head.ref_inv_id`

`ref_rtv_order_no` = `inv_head.ref_rtv_order_no`

`ref_price_change` = `inv_head.ref_price_change`

`ref_rsn_code` = `inv_head.ref_rsn_code`

`terms` = `inv_head.terms`

`due_date` = `inv_head.due_date`

`payment_method` = `inv_head.payment_method`

`terms_dscnt_pct` = `inv_head.terms_dscnt_pct`

`terms_dscnt_appl_ind` = `inv_head.terms_dscnt_appl_ind`

`freight_terms` = `inv_head.freight_terms`

`inv_date` = `inv_head.inv_date`

`force_pay_ind` = `inv_head.force_pay_ind`

`post_date` = today's date

`currency_code` = `inv_head.currency_code`

`exchange_rate` = `inv_head.exchange_rate`

`total_payment_merch_cost` = `inv_head.total_merch_cost`

`total_payment_qty` = `inv_head.total_qty`

`comments` = `inv_head.comments`

`addr_key` = `inv_head.addr_key`

payment_ref_no = invc_head.payment_ref_no

payment_date = invc_head.payment_date

proof_of_delivery_no = invc_head.proof_of_delivery_no

direct_ind = invc_head.direct_ind

For the insert into IIF_NON_MERCH, gather the following values for each invoice:

invc_id = invc_non_merch.invc_id

non_merch_code = invc_non_merch.non_merch_code

non_merch_amt = invc_non_merch.non_merch_amt

vat_code = invc_non_merch.vat_code

service_perf_ind = invc_non_merch.service_perf_ind

store = invc_non_merch.store

For the insert into IIF_MERCH_VAT, gather the following values for each invoice:

invc_id = invc_merch_vat.invc_id

vat_code = invc_merch_vat.vat_code

total_cost_excl_vat = invc_merch_vat.total_cost_excl_vat

For the insert into IIF_DETAIL, gather the following values for each invoice:

invc_id = invc_detail.invc_id

sku = invc_detail.sku

payment_unit_cost = invc_detail.invc_unit_cost

payment_qty = invc_detail.invc_qty

payment_vat_rate = invc_detail.invc_vat_rate

cost_dscrpnncy_ind = invc_detail.cost_dscrpnncy_ind

qty_dscrpnncy_ind = invc_detail.qty_dscrpnncy_ind

vat_dscrpnncy_ind = invc_detail.vat_dscrpnncy_ind

For the insert into IIF_MATCH_DETAIL, gather the following values for each invoice (invoice types 'C', 'D' and 'M' will not have records on this table):

invc_id = invc_match_wksht.invc_id

sku = invc_match_wksht.shipment

order_no = the order_no from SHIPMENT for invc_match_wksht.shipment

asn_no = the ext_shipment from SHIPMENT for invc_match_wksht.shipment

shipment = invc_match_wksht.shipment

rcpt_date = the receive_date from SHIPMENT for invc_match_wksht.shipment

For invoices matched at the totals level, where there are no invc_match_wksht records, the values should be gathered from shipsku instead.

Once all IIF records are inserted into the array for the invoice, update the invoice and receipt header statuses appropriately.

Set the post_date on INVC_HEAD to today's date.

If the action taken for the invoice is POST, set the status on INVC_HEAD to 'P' (posted).

If the invoice was in 'A' status, check each shipment matched to the invoice (retrieve using match_invc_id on SHIPSKU) to see if it is matched to any other invoices not in 'P' status. If it is, or if there are still SKUs on the shipment that are unmatched (match_invc_id is NULL), leave the shipment's invc_match_status as it is. If it is not, and all SKUs on the shipment are matched, set the shipment's invc_match_status to 'C'.

End LOOP.

Table to Table

Source Table	Source Column	Field Type
INVC_DISCOUNT	invc_id	number(10)
	Seq_no	number(6)
	discount_type	varchar2(6)
	discount_value	number(20,4)
	applies_to_amt	number(20,4)

Target Table	Target Column	Calculations
IIF_DISCOUNT	Invc_id	None
	seq_no	None
	discount_type	None
	discount_value	None
	applies_to_amt	None

process_invc()

after the calls to insert_iif_non_merch() and insert_iif_merch_vat(), and before the call to insert_iif_detail(), make a call to the new function insert_iif_discount()

insert_iif_discount()

perform a select insert to move all data from INVC_DISCOUNT to IIF_DISCOUNT for the invoice in the pa_invc_array current record

Technical Issues

N/A

Automatic Invoice Matching [invmatch]

Design Overview

This batch program will perform the automatic invoice matching process. For each invoice on the INVC_MATCH_QUEUE table, a summary match will be attempted followed by a detail-level match (if invoice details are available). Successfully matched invoices will be set to Matched status and automatically approved if from automatic approval suppliers. Invoices that fail automatic matching will have their match fail indicator set to 'Y'. On successful completion of this program, all records will be deleted from the INVC_MATCH_QUEUE table.

Scheduling Constraints

Processing Cycle:	daily
Scheduling Diagram:	after ediupinv, rcvupld, before invpost
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	supplier

Restart Recovery

The logical unit of work for the automatic invoice matching module will be the invoice transaction. Restart/recovery will be based on the invoice transaction. The commit_max_ctr field on the restart_control table will determine the number of transactions that equal a logical unit of work.

The commit_max_ctr field should be set to prevent excessive rollback space usage. The recommended commit counter setting is 10000 records (subject to change based on experimentation).

Program Flow

N/A

Shared Modules

INVC_MATCH_SQL.TOTAL_MATCH – to summary match a given invoice.

INVC_MATCH_SQL.ITEM_MATCH_ALL – to detail match a given invoice.

Driving Cursor

```
SELECT imq.invc_id,
       ih.status
FROM   invc_head ih,
       invc_match_queue imq,
       v_restart_supplier vrs
WHERE  imq.invc_id = ih.invc_id
       AND ih.supplier = vrs.driver_value
       AND vrs.driver_name = :os_restart_driver_name
       AND vrs.num_threads = :oi_restart_num_threads
       AND vrs.thread_val = :oi_restart_thread_val
       AND vrs.driver_value = ih.supplier
       AND (ih.invc_id > NVL(:os_restart_invc_id, -999))
ORDER BY ih.invc_id;
```

Function Level Description

init():

Restart/recovery initialization; size arrays for driving cursor to fetch into

process():

In a LOOP, fetch the driving cursor using an array. For each invoice fetched, attempt to match it as follows:

Check the invoice's status on INVC_HEAD. If the status is not 'U' or 'R', the invoice is already matched. Continue processing with the next invoice (apart from the last step of deleting off INVC_MATCH_QUEUE).

Call the function `check_header_match`. This will do the following:

1. call `INVC_MATCH_SQL.TOTAL_MATCH` to attempt a totals match of the invoice.
2. If `O_reconciled` is `FALSE`, the invoice cannot be matched. The `match_fail_ind` on `INVC_HEAD` should be updated to 'Y' and processing should continue on the next invoice (apart from the last step of deleting off `INVC_MATCH_QUEUE`).
3. If `O_match` is `TRUE`, the invoice is matched and processing can continue on the next invoice (apart from the last step of deleting off `INVC_MATCH_QUEUE`).
4. If `O_match` is `FALSE`, line item matching will be attempted if line item details exist on `INVC_DETAIL`. Call the function `INVC_MATCH_SQL.ITEM_MATCH_ALL` to attempt line item matches of the invoice.
5. If `O_match` is `TRUE` from `ITEM_MATCH_ALL`, the invoice is matched and processing can continue on the next invoice (apart from the last step of deleting off `INVC_MATCH_QUEUE`).
6. If `O_match` is `FALSE` from `ITEM_MATCH_ALL`, the invoice is not matched. The `match_fail_ind` on `INVC_HEAD` should be updated to 'Y'.

Once an invoice is processed, delete its record off `INVC_MATCH_QUEUE`.

End LOOP.

`check_header_match(ls_inv_id):`

Call `INVC_MATCH_SQL.TOTAL_MATCH`; if reconciled but not matched, call `INVC_MATCH_SQL.ITEM_MATCH_ALL`. (see detailed description above).

`final():`

restart/recovery close

Technical Issues

N/A

[invprg.pc]

Design Overview

This program will purge old (older than order_history_months) posted invoices that have not already been purged by ordprg.pc (which purges invoices associated with an order). This includes all types of invoices—non-merchandise, credit notes, credit note requests, debit memos, and consignment invoices. Regular merchandise invoices will primarily be deleted through ordprg.pc but will be deleted by invprg.pc if they still exist in the system. The age of the invoices will be determined from the match date; if there is no match date, the invoice date will be used.

Scheduling Constraints

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
UNIT_OPTIONS	No	Yes	No	No	No
PERIOD	No	Yes	No	No	No
INVC_HEAD	Yes	Yes	No	No	Yes
INVC_DETAIL	Yes	No	No	No	Yes
INVC_XREF	Yes	No	No	No	Yes
INVC_MATCH_QUEUE	Yes	No	No	No	Yes
INVC_NON_MERCH	Yes	No	No	No	Yes
INVC_MERCH_VAT	Yes	No	No	No	Yes
INVC_MATCH_WKSHT	Yes	No	No	No	Yes

Processing Cycle: Ad hoc (monthly)

Scheduling Diagram: This program should run after ordprg.pc (Invoices that are matched to an order are purged by ordprg.pc)

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: None

Restart Recovery

The logical unit of work is the invoice. A commit should occurs every 2500 records (subject to experimentation)

Driving cursor:

```

SELECT invc_id

      FROM invc_head

WHERE status = 'P'

      AND
MONTHS_BETWEEN(TO_DATE(:os_vdate,'YYYYMMDD'),NVL(match_date, invc_date))

>:oi_hist_months

      AND not exists (SELECT 'x'

                      FROM sa_tran_head s

                      WHERE i.payment_date = to_date (to_char (s.tran_datetime,
'YYYYMMDD'), 'YYYYMMDD')

                      AND (i.ext_ref_no = s.vendor_inc_no

                          OR i.ext_ref_no = s.payment_ref_no

                          OR i.ext_ref_no = s.proof_of_delivery_no))

ORDER BY i.invc_id;
```

Shared Modules

INVC_SQL.DELETE_INVC—for a given invoice, delete from all associated invoice tables

Function Level Description

init()

Fetch vdate from period and order_history_months from unit_options.

Restart/recovery initialization.

Call size_array to set up memory for the array the driving cursor will fetch into.

process()

In a loop:

Array fetch driving cursor (get invoices to delete)

Call delete_invoices to do actual deletion for each invoice in the array.

Call restart_commit after the entire array has been processed, making sure to force a commit.

delete_invoices()

Call the invc_sql.delete_inv function to delete invoices. The SKU parameter should be passed in as NULL so that the entire invoice will be deleted.

size_array()

Allocate memory for invoice id array.

final()

Restart/recovery close.

I/O Specification

N/A

Item Download [itemdnld]

Design Overview

This program downloads item, upc and pack component information to the warehouse. Triggers on the item tables will fire when information has been added, modified, or deleted from the specified tables. When the trigger fires, it will write the item, upc, upc_supplement, action type and packsku (if the bill of materials (BOM) of a pack item has been modified) to the item staging table, lif_item. This program will then read from the staging table and write item, BOM and upc download files, for the warehouse to process. The item download to RDM needs to be send all new or deleted UPC codes associated to an item to the UPC file.

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: itemdnld does not have any scheduling constraints. It can run multiple times during a batch schedule if needed. The rms2rls.awk file conversion program should run after this program to properly interface with Nautilus.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

The Logical Unit of Work for the program will be a single row from the staging table, lif_item. Restart/Recovery will be used for init and commit.

```
SELECT li.item,
       li.upc,
       li.upc_supplement,
       li.packsku,
       li.action_type,
       li.rowid,
       '||TO_CHAR(li.item)
FROM lif_item li
WHERE item > NVL(:os_restart_sku, -999)
ORDER BY item, upc;
```

Program Flow

The program will select the fields from the staging table, lif_item and write the records to the flat file in the standard format. Records which were successfully written to the flat file will be deleted from lif_item, while records which failed to will not be deleted from lif_item.

Init()

Initialize restart recovery

Open output files (item, upc and bill of materials (BOM))

Write FHEAD record.

Process()

Open the driving cursor in a for loop.

Fetch the data.

If action_type in ('A', 'M')

If the packsku field and the upc fields are both null on lif_item table

Open details cursor, which will retrieve the primary upc code, the primary supplier, sku description, dept, class, and system indicator for the file details. If the supplier pack size is 1, then send the actual dimension and weight. If the supplier pack size is not 1, then send NULL

If the action_type = 'M', then send NULL for dimension and weight, unit retail, and ticket qty.

Write Item FDETL record

If the packsku field on the lif_item table is not null, then a BOM file will be created

Get the component quantity from the v_packsku_qty table for the pack item and component values found from the main driving cursor,

Write BOM FDETL record to file with the appropriate action type.

If the action_type = 'D'

If packsku field is NULL, which indicates that a component may have been deleted from a pack.

Write Item FDETL record with action type = 'D' and item.

If packsku field is not NULL

Check if the sku is being entirely deleted from the pack or if the sku is being deleted from one template or style that makes up the pack. If the sku_qty retrieved from v_packsku_qty is zero (not found), then the entire item is being deleted from the pack, therefore, action type = 'D'. If the sku_qty is greater than zero (found), then the action type = 'M'.

Write BOM FDETL record with appropriate action type, pack item, and packsku.

If the upc field on the lif_item table is not null, then a UPC file will be created.

Write UPC FDETL record to file with the appropriate action type.

If action_type = 'A'

If upc field is NOT NULL write out a new UPC file.

If the action_type = 'D'

If upc field is NOT NULL write out a new UPC delete file.

Check_catch_weight()

This function is needed to determine whether the item is a catch_weight item or not. This function should also determine the standard_unit_weight of the item.

This function should be called in Process() after c_get_item is fetched and before information is written to item_file.

An item is catch_weight when its UOM is anything other than 'EA'. In order to determine the UOM of an item, the following columns need to be used:

RAG_STYLE.STANDARD_UOM for fashion items,
WIN_SKUS.STANDARD_UOM for staple items and
PACKHEAD.STANDARD_UOM for pack items.

(DESC_LOOK.SYSTEM_IND will display 'F' for fashion item, 'S' for staple item and 'P' for pack item.)

Once an item is found out to be catch_weight, then catch_weight field should be populated with 'Y'. If the item is not catch_weight (UOM = 'EA') then catch_weight field should be 'N'.

The standard_unit_weight field will hold the standard unit of weight equivalent to one 'EACH', meaning if an item is catch_weight then standard unit = (1/conversion factor) where conversion factor =(EACH/UOM). Conversion factors can be found in RAG_STYLE, WIN_SKUS and PAKCHEAD tables for respective item and UOM.

After catch_weight and standard_unit_weight are found, these values should be outputted to item_file.

Win_skus_uom()

Fetch UOM from win_skus table.

Rag_style_uom()

Fetch UOM from rag_style table.

Packhead_uom()

Fetch UOM from packhead table.

final()

Write the FTAIL record.

Commit if no error, else rollback

Function Level Description

N/A

I/O Specification

Input File

The input file should be accepted as a runtime parameter at the command line.

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	Ten zeroes: 0000000000	ID of current line being processed by input file.
	File Type Definition	Char(4)	ITEM	Identifies file as 'Vendor download'
	File Create Date	Date	create date	date file was written by external system
Trans-action Detail	File Type Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	Action Type	Char(1)	lif_sup. action_type	
	Location (DC)	Char(4)	NULL	Warehouse location. This field is not used.
	Transaction Date/Time	Char (12) format: YYYYMMDDH HMI	sysdate	sysdate without the seconds
	Item ID	Char (16)	lif_item.sku	Valid item identifier for a staple SKU, fashion SKU, or Pack Item
	Product Description	Char (60)	Desc_look.desc_up	SKU description for each Fashion SKU, Staple SKU, or Pack Item.

Record Name	Field Name	Field Type	Default Value	Description
	Vendor Number	Char (7)	Item_supplier. Supplier	primary supplier of the item.
	Vendor Style	Char (15)	rag_skus.style	Vendor Style associated with the item sent from the vendor.
	Item Length	Number(6)	Win_skus/rag_style.ship _carton_length (if pack size = 1) 1	Item Length . If case pack size = 1, then send actual carton dimension, otherwise sent 1as a default. When modifying this value should be NULL.
	Item Width	Number(6)	Win_skus/rag_style.ship _carton_width (if pack size = 1) 1	Item Width. If case pack size = 1, then send actual carton dimension, otherwise sent 1as a default. When modifying this value should be NULL.
	Item Height	Number(6)	Win_skus/rag_style.ship _carton_height (if pack size = 1) 1	Item Height. If case pack size = 1, then send actual carton dimension, otherwise sent 1as a default. When modifying this value should be NULL.

Record Name	Field Name	Field Type	Default Value	Description
	Item Weight	Number(8)	Item_supplier.ship_Carton_wt / item_supplier.suppack_size	Item Weight. When modifying this value should be NULL.
	Unit Ticket Qty	Number (3)	1	Number of tickets per unit
	Velocity	Number (2)	N/A	Measure of turn time through DC
	Item UPC	Char (16)	Upc_supplier.upc	Primary UPC for the primary supplier
	High Value Indicator (Y or N)	Char (1)	N/A	Whether the item is high value
	Hazardous Material Code	Char (6)	N/A	Hazardous material code
	Department	Number (4)	desc_look.dept	Department the sku belongs to.
	Standard Unit Qty	Number (8)	NULL	Standard unit quantity for an item
	Container Type	Char (6)	NULL	Preferred container type
	Preferred Zone	Char (2)	NULL	Preferred storage zone for item
	Primary Location Type	Char (6)	NULL	Primary location type
	Secondary Location Type	Char (6)	NULL	Secondary Location Type
	Tertiary Location Type	Char (6)	NULL	Tertiary Location Type
	Class	Char (7)	desc_look.class	Class of merchandise

Record Name	Field Name	Field Type	Default Value	Description
	Kitting WIP Code	Char (6)	Code_detail.code	WIPP code from codes table
	Ticket Type	Number (4)	Zero if ticket_type_id is NULL on item_ticket	Default type of ticket.
	Retail Price	Number (8)	Item_zone_price.unit_retail	Base unit retail for item
	Color	Char(24)	colour.colour_desc	Fashion sku's color
	Size	Char(6)	rag_skus.size1	Fashion sku's size1
	Fit	Char(6)	rag_skus.size2	Fashion sku's size2, if applicable
	Shade	Char(6)	NULL	Shade of the Item.
	Unit_pick_system_code	Char(4)	NULL	Not captured in RMS – send NULL.
	Roundable_flag	Char(1)	NULL	Not captured in RMS – send NULL.
	Catch_weight	Char(1)	NULL	If weight captured, catch_weight is 'Y', in another case this is 'N'.
	Std_unit_weight	Number (21)	NULL	If catch_weight = 'Y', this column has value. If catch_weight = 'N' – send NULL.
	Perishable_ind	Char(1)	NULL	Not captured in RMS – send NULL.

Record Name	Field Name	Field Type	Default Value	Description
	Division	Number (4)	Division.division	Division the sku belongs to.
	Std_casepack	Number (8)	Item_supp_country.suppl_pack_size	Standard pack size.
	Subclass	Number (4)	desc_look.subclass	Subclass the sku belongs to.
	Inner_pack_qty	Number (8)	Item_supp_country.inner_pack_size	Inner pack size
	Single_contain_bulk	Char(1)	NULL	Not captured in RMS – send NULL.
	Expiration_days	Number (4)	NULL	Not captured in RMS – send NULL.
File Trailer	File Type Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
	File Record Counter	Number(10)	Determined Internally	Number of records/transactions processed in current file (only records between head & tail)

Bill of Materials (BOM) Download:

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	Ten zeroes: 0000000000	ID of current line being processed by input file.

Record Name	Field Name	Field Type	Default Value	Description
	File Type Definition	Char(4)	BOMD	Identifies file as 'Vendor download'
	File Create Date	Date	create date	date file was written by external system
Transaction Detail	File Type Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
	Action Type	Char(1)	lif_item.action_type	A = add, M = modify, D = delete
	Location (DC)	Char(4)	NULL	Code for the DC (future use)
	Transaction Date/Time	Char (12) format: YYYYMMDDHHMI	sysdate Format: YYYYMMDDHHMI	Date/time created on host
	Master Item ID	Char(16)	lif_item.sku	Item identifier of the kit
	Component Item ID	Char(16)	Lif_item.packsku	Item identifier of the item
	Unit Qty	Number(8)	Packsku.sku_qty	Quantity of component item id's included in the master item id
File Trailer	File Type Descriptor	Char(5)	FTAIL	Identifies file record type

Record Name	Field Name	Field Type	Default Value	Description
	File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
	File Record Counter	Number(10)	Determined internally	Number of records/transactions processed in current file (only records between head & tail)

UPC File

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	Numeric ID(10)	Sequential number Created by program.	ID of current line being created for output file.
	File Type Definition	Char(4)	UPCD	Identifies file as 'UPC Download'
	File Create Date	Date	Create date	YYYYMMDD HHMM
	File Create Date	Date	Create date	D
Transaction Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being created for output file.
	Action Type	Char(1)	Lif_item.action_type	A = add, D = delete
	Location (DC)	Char(4)	NULL	Code for the DC (future use)

Record Name	Field Name	Field Type	Default Value	Description
	Transaction Date/Time	Char (12) format: YYYYMMDD HHMI	sysdate Format:	Date/time created on host
	Master Item ID	Char(16)	Lif_item.sku	Valid item identifier for a staple SKU, fashion SKU, or Pack Item
	UPC ID	Char(16)	Lif_item.upc	UPC identifier of the item
	UPC Supplement	Number(5)	Upc_ean.upc_supplement	Used along with the UPC field to identify the Universal Product Code.
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Numeric ID(10)	Sequential number Created by program.	ID of current line being created for output file.
	File Record Counter	Number(10)	Determined Internally	Number of records/transactions processed in current file (only records between head & tail)

Note that there is a space between fields in the RLS flat file format

Note that both truncation and padding is required to properly interface with Nautilus system. In the case of padding, use the RMS field type to determining Left or Right padding (numeric: left, text right)

Error File

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

N/A

Bill of Lading Purge [ladprg]

Design Overview

The purpose of this module is to purge bill of lading files run by deleting all rows in the lading head and detail tables. The purge condition is determined by the number of months of lading history to be retained and the date the bill of lading was printed.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
LADHEAD	No	Yes	No	No	Yes
LADDETAIL	No	No	No	No	Yes
PERIOD	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE AD-HOC (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

This function deletes lad_head and lad_detail records that are older than the number of months specified by the system option bill_of_lading_history_mths.

I/O Specification

N/A

Technical Issues

N/A

Layaway Purge [layprg]

Design Overview

The purpose of this module is to purge layaway accounts that have exceeded the retention period from the database. The purge condition is determined by the number of months of layaway history to be retained in the System Options and the following conditions:

Only those layaway accounts with status of paid in full, refunded, or returned to stock that contain a sale date older than twice the number of days allowed for completion of a layaway contract are processed.

The processing varies depending on the status:

- Paid in full: The customer's layaway account and all associated item and payment rows are deleted from the database.
- Refunded: All service fees are accumulated and used at the end of the program to reduce the forfeit and total layaway amounts. The customer's layaway account and all associated item and payment rows are then deleted from the database.
- Returned to stock: The layaway account details are printed for archiving, all service fees and unclaimed payments are accumulated and used at the end of the program to reduce the forfeit, unclaimed, and total layaway amounts. The customer's layaway account and all associated item and payment rows are then deleted from the database.

When all applicable layaway accounts have been processed, the current layaway total is created or updated by reducing the forfeit and unclaimed values by the amounts accumulated from refunded or returned to stock accounts. These accumulated totals are printed on the report to allow reconciliation and posting to the general ledger.

Tables affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
CLASS	No	Yes	No	No	No
DEPS	No	Yes	No	No	No
LAY_CUST	No	Yes	No	No	Yes
LAY_ITEMS	Yes	Yes	No	No	Yes
LAY_PAY	Yes	Yes	No	No	Yes
LAY_TOTAL	No	Yes	Yes	Yes	No
PERIOD	No	Yes	No	No	No
SUBCLASS	No	Yes	No	No	No
STORE	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
WH	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE AD-HOC (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

Process()

This function checks the status of existing layaway accounts. It then deletes layaway accounts with a status of paid in full, refunded, or returned to stock that have exceeded the retention period from the LAY_CUST, LAY_ITEMS, and LAY_PAY tables and adjusts the forfeit, unclaimed, and total layaway amounts.

I/O Specification

N/A

Technical Issues

N/A

Letter of Credit Download [lcadnld]

Design Overview

The lcadnld program is used to process Letter of Credit (LC) applications, in order to be sent to the bank, based on the letter of credits flagged during the day by the Select LC Applications action on the LC Find form and written to the staging table LC_DOWNLOAD. This program has one output file which contains all records for all LC applications. This program should be included in the standard download phase of the batch cycle. The individual LCs flagged for a given day will be downloaded in full, with application and detail information written to a Retek standardized flat file. Another program will format the information from this file into individualized bank flat files using a S.W.I.F.T. format (SWIFT formats are used when transmitting financial information and are a globally recognized set of standards). After this program has processed all of the applications on LC_DOWNLOAD, these records will be deleted and the LCs will have their status changed to 'E'xtracted on LC_HEAD.

Affected Tables:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
LC_HEAD	Yes	Yes	No	Yes	No
LC_DETAILS	Yes	Yes	No	No	No
LC_DOWNLOAD	No	Yes	No	No	Yes
SUP_IMPORT_ATTR	No	Yes	No	No	No
ADDR	Yes	Yes	No	No	No
PARTNER	Yes	Yes	No	No	No
SUPS	Yes	Yes	No	No	No
CODE_DETAIL	Yes	Yes	No	No	No
RAG_SKUS	Yes	Yes	No	No	No
WIN_SKUS	Yes	Yes	No	No	No
DOC	No	Yes	No	No	No
OUTLOC	Yes	Yes	No	No	No
ITEM_IMPORT_ATTR	No	Yes	No	No	No
REQ_DOC	Yes	Yes	No	No	No

Scheduling Constraints

Processing Cycle:	PHASE 4 (daily)
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	Perl script lcmt700.
Threading Scheme:	No threading due to low volume.

Driving Cursor

The cursor includes three parts. The first part joins to the rag_skus table and selects the style where the rag_skus.sku is equal to lc_detail.item, a group by is used to sum the qty. The second part joins win_skus and selects the sku where win_skus.sku is equal to lc_detail.itm or sku is NULL. The third part joins packhead and selects the pack_no where the packhead.pack_no is equal to lc_detail.item.

```
SELECT ROWIDTOCHAR(h.rowid),  
       ROWIDTOCHAR(l.rowid),  
       h.issuing_bank,  
       NVL(h.advising_bank, ' '),  
       h.lc_ref_id,  
       NVL(h.bank_lc_id, ''),  
       h.form_type,  
       h.lc_type,  
       s.revocable_ind,  
       h.transferable_ind,  
       TO_CHAR(h.application_date, 'YYYYMMDDHH24MISS'),  
       TO_CHAR(h.expiration_date, 'YYYYMMDDHH24MISS'),  
       h.place_of_expiry,  
       h.applicant,  
       h.beneficiary,
```


h.currency_code,
h.exchange_rate * :ld_rate_dec,
NVL(h.origin_country_id, ''),
h.presentation_terms,
h.purchase_type,
NVL(h.advice_method, ''),
NVL(h.issuance, ''),
h.amount_type,
h.amount * :ll_no_dec,
h.variance_pct * :ll_no_dec,
NVL(h.specification, ''),
NVL(h.credit_avail_with, ''),
NVL(h.drafts_at, ''),
NVL(h.paying_bank, ''),
NVL(h.negotiating_bank, ''),
NVL(h.confirming_bank, ''),
NVL(h.transferring_bank, ''),
h.partial_shipment_ind,
h.transshipment_ind,
NVL(h.fob_title_pass, ''),
NVL(h.fob_title_pass_desc, ''),
h.with_recourse_ind,
TO_CHAR(h.latest_ship_date, 'YYYYMMDDHH24MISS'),
TO_CHAR(h.earliest_ship_date, 'YYYYMMDDHH24MISS'),
NVL(TO_CHAR(h.lc_neg_days), ''),
NVL(h.transport_to, ''),

```
        ol.outloc_desc,
        h.comments,
        d.order_no,
        TO_CHAR(rs.style),
        MAX(NVL(d.cost, 0) * :ll_no_dec),
        SUM(NVL(d.qty, 0) * :ll_no_dec),
        MIN(NVL(TO_CHAR(d.earliest_ship_date,
'YYYYMMDDHH24MISS'), ' ')),
        MAX(NVL(TO_CHAR(d.latest_ship_date,
'YYYYMMDDHH24MISS'), ' ')),
        d.merch_desc,
        d.comments,
        rst.standard_uom,
        dl.desc_up
FROM lc_head h,
     lc_detail d,
     rag_skus rs,
     sup_import_attr s,
     lc_download l,
     outloc ol,
     desc_look dl,
     rag_style rst
WHERE h.lc_ref_id = l.lc_ref_id
     AND d.lc_ref_id = l.lc_ref_id
     AND s.supplier = h.beneficiary
     AND rs.sku = d.item
     AND ol.outloc_id = h.transport_to
```

```
AND l.lc_ref_id > nvl(:ps_restart_lc,0)
AND l.lc_download_type = 'A',
AND dl.sku = rst.style
AND rst.style = rs.style
GROUP BY h.rowid,
        l.rowid,
        h.issuing_bank,
        h.advising_bank,
        h.lc_ref_id,
        h.bank_lc_id,
        h.form_type,
        h.lc_type,
        s.revocable_ind,
        h.transferable_ind,
        h.application_date,
        h.expiration_date,
        h.place_of_expiry,
        h.applicant,
        h.beneficiary,
        h.currency_code,
        h.exchange_rate,
        h.origin_country_id,
        h.presentation_terms,
        h.purchase_type,
        h.advice_method,
        h.issuance,
```

h.amount_type,
h.amount,
h.variance_pct,
h.specification,
h.credit_avail_with,
h.drafts_at,
h.paying_bank,
h.negotiating_bank,
h.confirming_bank,
h.transferring_bank,
h.partial_shipment_ind,
h.transshipment_ind,
h.fob_title_pass,
h.fob_title_pass_desc,
h.with_recourse_ind,
h.latest_ship_date,
h.earliest_ship_date,
h.lc_neg_days,
h.transport_to,
ol.outloc_desc,
h.comments,
d.order_no,
rs.style,
d.merch_desc,
d.comments,
rst.standard_uom,

```
        dl.desc_up
UNION ALL
SELECT ROWIDTOCHAR(h2.rowid),
       ROWIDTOCHAR(l2.rowid),
       h2.issuing_bank,
       NVL(h2.advising_bank, ' '),
       h2.lc_ref_id,
       NVL(h2.bank_lc_id, ' '),
       h2.form_type,
       h2.lc_type,
       s2.revocable_ind,
       h2.transferable_ind,
       TO_CHAR(h2.application_date, 'YYYYMMDDHH24MISS'),
       TO_CHAR(h2.expiration_date, 'YYYYMMDDHH24MISS'),
       h2.place_of_expiry,
       h2.applicant,
       h2.beneficiary,
       h2.currency_code,
       h2.exchange_rate * :ld_rate_dec,
       NVL(h2.origin_country_id, ' '),
       h2.presentation_terms,
       h2.purchase_type,
       NVL(h2.advice_method, ' '),
       NVL(h2.issuance, ' '),
       h2.amount_type,
       h2.amount * :ll_no_dec,
```

```
h2.variance_pct * :ll_no_dec,
NVL(h2.specification, ''),
NVL(h2.credit_avail_with, ''),
NVL(h2.drafts_at, ''),
NVL(h2.paying_bank, ''),
NVL(h2.negotiating_bank, ''),
NVL(h2.confirming_bank, ''),
NVL(h2.transferring_bank, ''),
h2.partial_shipment_ind,
h2.transshipment_ind,
NVL(h2.fob_title_pass, ''),
NVL(h2.fob_title_pass_desc, ''),
h2.with_recourse_ind,
TO_CHAR(h2.latest_ship_date, 'YYYYMMDDHH24MISS'),
TO_CHAR(h2.earliest_ship_date, 'YYYYMMDDHH24MISS'),
NVL(TO_CHAR(h2.lc_neg_days), ''),
NVL(h2.transport_to, ''),
ol2.outloc_desc,
h2.comments,
d2.order_no,
TO_CHAR(d2.item),
NVL(d2.cost, 0) * :ll_no_dec,
NVL(d2.qty, 0) * :ll_no_dec,
NVL(TO_CHAR(d2.earliest_ship_date, 'YYYYMMDDHH24MISS'), '
'),
NVL(TO_CHAR(d2.latest_ship_date, 'YYYYMMDDHH24MISS'), ''),
d2.merch_desc,
```

```
d2.comments,
ws.standard_uom,
dl2.desc_up
FROM lc_head h2,
lc_detail d2,
win_skus ws,
sup_import_attr s2,
lc_download l2,
outloc ol2,
desc_look dl2
WHERE h2.lc_ref_id = l2.lc_ref_id
AND dl2.system_ind = 'S'
AND d2.lc_ref_id = l2.lc_ref_id
AND s2.supplier = h2.beneficiary
AND (ws.sku = d2.item
      OR (d2.item is NULL
          AND ws.sku = (select MIN(sku) from win_skus)))
AND ol2.outloc_id = h2.transport_to
AND l2.lc_ref_id > nvl(:ps_restart_lc,0)
AND l2.lc_download_type = 'A'
AND ((dl2.sku = d2.item AND d2.item is NOT NULL)
      OR (d2.item is NULL AND dl2.sku = ws.sku))
UNION ALL
SELECT ROWIDTOCHAR(h3.rowid),
       ROWIDTOCHAR(l3.rowid),
       h3.issuing_bank,
```

NVL(h3.advising_bank, ' '),
h3.lc_ref_id,
NVL(h3.bank_lc_id, ' '),
h3.form_type,
h3.lc_type,
s3.revocable_ind,
h3.transferable_ind,
TO_CHAR(h3.application_date, 'YYYYMMDDHH24MISS'),
TO_CHAR(h3.expiration_date, 'YYYYMMDDHH24MISS'),
h3.place_of_expiry,
h3.applicant,
h3.beneficiary,
h3.currency_code,
h3.exchange_rate * :ld_rate_dec,
NVL(h3.origin_country_id, ' '),
h3.presentation_terms,
h3.purchase_type,
NVL(h3.advice_method, ' '),
NVL(h3.issuance, ' '),
h3.amount_type,
h3.amount * :ll_no_dec,
h3.variance_pct * :ll_no_dec,
NVL(h3.specification, ' '),
NVL(h3.credit_avail_with, ' '),
NVL(h3.drafts_at, ' '),
NVL(h3.paying_bank, ' '),

```

        NVL(h3.negotiating_bank, ' '),
        NVL(h3.confirming_bank, ' '),
        NVL(h3.transferring_bank, ' '),
        h3.partial_shipment_ind,
        h3.transshipment_ind,
        NVL(h3.fob_title_pass, ' '),
        NVL(h3.fob_title_pass_desc, ' '),
        h3.with_recourse_ind,
        TO_CHAR(h3.latest_ship_date, 'YYYYMMDDHH24MISS'),
        TO_CHAR(h3.earliest_ship_date, 'YYYYMMDDHH24MISS'),
        NVL(TO_CHAR(h3.lc_neg_days), ' '),
        NVL(h3.transport_to, ' '),
        ol3.outloc_desc,
        h3.comments,
        d3.order_no,
        TO_CHAR(d3.item),
        NVL(d3.cost, 0) * :ll_no_dec,
        NVL(d3.qty, 0) * :ll_no_dec,
        NVL(TO_CHAR(d3.earliest_ship_date, 'YYYYMMDDHH24MISS'), '
    '),
        NVL(TO_CHAR(d3.latest_ship_date, 'YYYYMMDDHH24MISS'), ' '),
        d3.merch_desc,
        d3.comments,
        ph.standard_uom,
        dl3.desc_up
FROM lc_head h3,
     lc_detail d3,

```

```
    packhead ph,  
    sup_import_attr s3,  
    lc_download l3,  
    outloc ol3,  
    desc_look dl3  
WHERE h3.lc_ref_id = l3.lc_ref_id  
      AND dl3.system_ind = 'P'  
      AND d3.lc_ref_id = l3.lc_ref_id  
      AND s3.supplier = h3.beneficiary  
      AND ph.pack_no = d3.item  
      AND ol3.outloc_id = h3.transport_to  
      AND l3.lc_ref_id > nvl(:ps_restart_lc,0)  
      AND l3.lc_download_type = 'A'  
      AND (dl3.sku = d3.item AND d3.item is NOT NULL)  
ORDER BY 3, 5, 44;
```

Restart Recovery

Restart/recovery for this program is set up at the lc_ref_id level.

The recommended commit counter setting is 1000 records (subject to change based on experimentation).

Program Flow



Shared Modules

LANGUAGE_SQL.GET_CODE_DESC

Function Level Description

Init():

This function performs preliminary processing and populates global variables. It will call `retek_init()` to start up restart-recovery and bring back the bookmark string in case of a restart. It will also populate `vdate`, to be used when writing to the output file. Then call `write_fhead()` to open the output file and write the FHEAD information.

Process():

This function drives the processing of the program. The driving cursor is fetched here which retrieves all the records from `lc_download` where the `lc_download.lc_ref_lc_id` value is greater than zero and `lc_download.lc_download_type='A'`. (The cursor includes two parts, the first part join the `rag_skus` and select the style which `rag_skus.sku=lc_detail.item`, the second part joins the `win_skus` and select the sku which `win_skus.sku=lc_detail.item` and `lc_detail.item` is null.)

Once the records are fetched, it sets the variance_pct field = ' ' if the amount_type is 'E'xact. It then calls write_thead() for every lc_ref_id. If lc_head.comments has been populated, then call write_thdcm(), to write the comments field to the output file. Then pass in the lc_ref_id to write_tdocs() to write a line for each description of goods and services, required document or additional condition associated with the LC. Then go on to call write_tdetl(), passing in the application_struct. If the lc_detail.merch_desc field has been populated, then call write_tmerc(), to write the merchandise description field to the output file. If the lc_detail.comments field has been populated, then call write_tdtcm(), to write the comments field to the output file. Finally, call write_ttail() if the lc_ref_id has changed (to ensure that the transaction tail is only called when all the driving cursor records for an individual LC have been processed). For every loop call update_lc_head() to update all the statuses on lc_head table and call delete_lc_download() to delete all the records on lc_download table. Restart/Recovery and committing of records is also performed here.

Final():

This function will call write_ftail() first to write the FTAIL information, then call retek_close() to wrap up restart-recovery and populate the final error message, if necessary.

Write_fhead():

This function will write the FHEAD line of the output file.

Write_thead():

This function, which takes in the application_struct, will write the THEAD line of the output file according to the layout presented later in this document. Most of the variables will be fetch from the driving cursor, and can be written to the file as they are, the others will call get_all_desc() and get_all_address().

Write_thdcm():

This function, which takes in the lc_head.comments, will write the THDCM line of the output file. Note: if the lc_head.comments field is not populated, then this function will not be called.

Write_tdocs():

This function, which will take in an individual `lc_ref_id`, will do a join between the `REQ_DOC` and the `DOC` table where the `REQ_DOC.DOC_ID = DOC.DOC_ID`, the `REQ_DOC.MODULE = 'LC'`, and the `REQ_DOC.KEY_VALUE_1 = lc_ref_id`. Fetch back the `swift_tag`, the `doc_id` and `doc_text` in a loop, writing an individual TDOC record for each `doc_text` fetched back. To explain further, each required document associated with an LC (and there can be more than one) will also have a `swift_tag` associated with it. This `swift_tag` is related to internationally recognized file format layouts that are used for financial transactions, often between banks. This program downloads the `swift_tag` because it corresponds to fields (ex. field 46A – Required Documents on form MT 700) on the SWIFT formatted file and will later be used by the scripts that translate this file into the bank required SWIFT format.

`Write_tdetl()`:

This function, which takes in the `application_struct`, will write the TDETL line of the output file according to the layout presented later in this document.

`Write_tmrc()`:

This function, which takes in the `lc_detail.merch_desc`, will write the TMERC line of the output file. If the `lc_detail.merch_desc` field is not populated, then this function will not be called.

`Write_tdtcm()`:

This function, which takes in the `lc_detail.comments`, will write the THDCM line of the output file. If the `lc_detail.comments` field is not populated, then this function will not be called.

`Write_ttail()`:

This function will write the TTAIL line of the output file.

`Write_ftail()`:

This function will write the FTAIL line of the output file.

`Cerate_format_string()`:

This function formats the strings for the FHEAD, THEAD, THDCM, TDOCS, TDETL, TMERC, TDTCM, TTAIL and FTAIL records in the output file.

`Init_arrays()`:

This function initializes the size of the array used for the driving cursor fetch the size of the restart max counter on `restart_control`.

`Get_all_desc()`:

This function will call `get_desc()`, passing a code and a `code_type` and will get the description for that code.

Form_type	LCFT
Lc_type	LCTP
Place_of_expiry	LCPE
Presenattion_terms	LCPT
Purchase_type	PURT
Advice_method	LCAD
Issuance	LCIS
Amount_type	LCAT
Specification	LCSP
Drafts_at	LCDA
Fob_title_pass	FOBT

Get_desc():

This function takes in a code and a code_type and fetches the description for that code using LANGUAGE_SQL.GET_CODE_DESC and passes the returned O_code_desc value out of the function.

Get_all_addresses():

This function will call get_address(), taking in Issuing_bank, Advising_bank, Applicant, Beneficiary, Credit_avail_with, Paying_bank, Negotiating_bank, Confirming_bank, Transferring_bank and will fetch back name and address information.

For Bank and Applicant, PARTNER.PARTNER_DESC should be fetched back where the partner_type = 'BK' and the partner_id = the bank (for the banks or the credit_avail_with) or where the partner_type = 'AP' and the partner_id = applicant (for the applicant). Then all the address information (add_1, add_2, add_3, city, state, post, country_id) should be fetched off the addr table. Again, fetch this information using the module = 'PTNR', key_value_1 = 'BK' (for the banks or credit_avail_with) or 'AP' (for the Applicant), and key_value_2 = issuing_bank ID or the applicant ID.

For the Beneficiary, first the supplier name should be fetched back from the SUPS table where Beneficiary = Supplier. Then, off the ADDR table the address information (add_1, add_2, add_3, city, state, post, country_id) should be fetched where the module = 'SUPP' and key_value_1 = beneficiary ID (key_value_2 is NULL).

Get_address():

This function will be called by `get_all_addresses()` and will return the information of name and address.

`Update_lc_head()`:

This function will updates the status of the Letter of Credits to 'E'xtracted once they have been successfully processed. This is by rowid.

`Delete_lc_download()`

This function will deletes all records from `LC_DOWNLOAD` that have been fully processed. Done by rowid.

I/O Specification

Output file

The file will be formatted as fixed-width values with no delimiters or punctuation. A new-line character will indicate the end of each record.

It should be noted that the TDOCS lines are in no way associated with the TDETL lines. The TDETL lines are details relating to an individual LC and the TDOCS lines are documents associated with an individual LC.

The THDCM line is only present when the comment field at the header level on `LC_HEAD` is populated. There can be only one THDCM line per THEAD line.

The TDTCM line is only present when the comment field at the detail level on `LC_DETAILS` is populated. There can be only one TDTCM line per TDETL line.

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)	line number in file	ID of current line being created for output file.
	File Type Definition	Char(4)	LCAP	Identifies file as 'Letter of Credit Application'
	File Create Date	Char(14)	create date	current date, formatted to 'YYYYMMDDHH24MISS'

Record Name	Field Name	Field Type	Default Value	Description
File Detail	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	ID of current line being created for output file.
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Issuing Bank	Char(10)	lc_head.issuing_bank	Used to sort the LCs into individualized bank SWIFT formatted files (using another program) – bank where LC application is headed.
	Issuing Bank Name	Char(40)	partner.partner_desc	The description from the partner table where partner_id = issuing_bank and partner_type = 'BK'.
	Issuing Bank Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Issuing Bank Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Issuing Bank Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Issuing Bank City	Char(20)	addr.city	City bank located in
	Issuing Bank State	Char(3)	addr.state	State, if applicable, where bank located in.
	Issuing Bank Post Code	Char(10)	addr.post	Post code, if applicable, where bank located in.
	Issuing Bank Country	Char(3)	addr.country_id	Country bank located in.

Record Name	Field Name	Field Type	Default Value	Description
	Advising Bank	Char(10)	lc_head.advising_bank	Used to sort the LCs into individualized bank SWIFT formatted files (using another program) – bank where LC application is headed.
	Advising Bank Name	Char(40)	Partner.partner_desc	The description from the partner table where partner_id = advising_bank and partner_type = 'BK'.
	Advising Bank Address 1	Char(30)	Addr.add_1	Mandatory line of address.
	Advising Bank Address 2	Char(30)	Addr.add_2	Non-mandatory line of address (can be null).
	Advising Bank Address 3	Char(30)	Addr.add_3	Non-mandatory line of address (can be null).
	Advising Bank City	Char(20)	Addr.city	City bank located in
	Advising Bank State	Char(3)	Addr.state	State, if applicable, where bank located in.
	Advising Bank Post Code	Char(10)	Addr.post	Post code, if applicable, where bank located in.
	Advising Bank Country	Char(3)	Addr.country_id	Country bank located in.
	Letter of Credit	Number(8)	lc_head.lc_ref_id	The LC_REF_ID off the LC_HEAD table.
	Form Type	Char(6)	lc_head.form_type	The level of detail that the LC will send to the issuing bank.
	Form Type Description	Char(40)	code_detail.code_desc	Describes the form type: Long or Short.
	Letter of Credit Type	Char(6)	lc_head.lc_type	The type of LC that is being applied for.

Record Name	Field Name	Field Type	Default Value	Description
	Letter of Credit Type Description	Char(40)	code_detail.code_desc	Describes the LC type: Master, Normal, Revolving.
	Form of Letter of Credit – I	Char(1)	sup_import_attr.revocable_ind	The REVOCABLE_IND off the SUP_IMPORT_ATTR table.
	Form of Letter of Credit – II	Char(1)	lc_head.transferable_ind	Indicates if LC transferable.
	Application Date	Char(14)	lc_head.application_date	Date the LC is created within RTM/RMS, formatted to 'YYYYMMDDHH24MISS'.
	Expiration Date	Char(14)	lc_head.expiration_date	The date the LC expires, formatted to 'YYYYMMDDHH24MISS'.
	Place of Expiry	Char(6)	lc_head.place_of_expiry	Code for the place the LC will expire.
	Place of Expiry Description	Char(40)	desc is retrieved through a decode	The description of the place the LC will expire.
	Applicant	Char(10)	lc_head.applicant	Party on whose behalf the LC is being issued.
	Applicant Name	Char(40)	partner.partner_desc	The description from the partner table where partner_id = applicant and partner_type = 'AP'.
	Applicant Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Applicant Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Applicant Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).

Record Name	Field Name	Field Type	Default Value	Description
	Applicant City	Char(20)	addr.city	City applicant located in
	Applicant State	Char(3)	addr.state	State, if applicable, where applicant located in.
	Applicant Post Code	Char(10)	addr.post	Post code, if applicable, where applicant located in.
	Applicant Country	Char(3)	addr.country_id	Country applicant located in.
	Beneficiary	Number(10)	lc.head.beneficiary	Party in favor of which the LC is being issued.
	Beneficiary Name	Char(32)	sups.sup_name	Beneficiary (supplier) name from the SUPS table,.
	Beneficiary Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Beneficiary Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Beneficiary Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Beneficiary City	Char(20)	addr.city	City beneficiary located in
	Beneficiary State	Char(3)	addr.state	State, if applicable, where beneficiary located in.
	Beneficiary Post Code	Char(10)	addr.post	Post code, if applicable, where beneficiary located in.
	Beneficiary Country	Char(3)	addr.country_id	Country beneficiary located in.
	Currency Code	Char(3)	lc_head.currency_code	The country of origin for the orders on the LC.
	Exchange Rate	Number (20,10)	lc_head.exchange_rate	Exchange_rate to convert LC currency

Record Name	Field Name	Field Type	Default Value	Description
				to RMS currency.
	Origin Country ID	Char(3)	lc_head.origin_country_id	Origin country of the orders associated with the LC.
	Presentation Terms	Char(6)	lc_head.presentation_terms	Code for the terms of presentation.
	Presentation Terms Description	Char(40)	desc is retrieved through a decode	Description of the terms of presentation.
	Purchase Type	Char(6)	lc_head.purchase_type	Code for the purchase type.
	Purchase Type Description	Char(40)	desc is retrieved through a decode	Description of the purchase type.
	Advice Method	Char(6)	lc_head.advice_method	Code for the advice method.
	Advice Method Description	Char(40)	desc is retrieved through a decode	Description of the advice method (eg. Full Wire, Mail, etc).
	Issuance	Char(6)	lc_head.issuance	Code for the issuance.
	Issuance Description	Char(40)	desc is retrieved through a decode	Description of the issuance (eg. Cable, Telex, etc).
	Amount Type	Char(6)	lc_head.amount_type	If 'E'xact, then amount must be exat, if 'A'pproximate then amount can be within variance percent.
	Amount Type Description	Char(40)	desc is retrieved through a decode	Description of amount_type.
	Amount	Number (20,4)	lc_head.amount	The total amt of the Letter of Credit.
	Variance Percent	Number (12,4)	lc_head.variance_pct	Allowed currency variance percent for the LC.
	Specification	Char(6)	lc_head.specification	Code for any condition for the credit, e.g. "maximum", etc.

Record Name	Field Name	Field Type	Default Value	Description
	Specification Description	Char(40)	desc is retrieved through a decode	Description of condition for the credit, e.g. "maximum", etc.
	Credit Available With	Char(10)	lc_head.credit_avail_with	Code for bank with which credit is available.
	Credit With Bank Name	Char(40)	partner.partner_desc	The description from the partner table where partner_id = credit_avail_with and partner_type = 'BK'.
	Credit With Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Credit With Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Credit With Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Credit With City	Char(20)	addr.city	City creditor located in
	Credit With State	Char(3)	addr.state	State, if applicable, where creditor located in.
	Credit With Post Code	Char(10)	addr.post	Post code, if applicable, where creditor located in.
	Credit With Country	Char(3)	addr.country_id	Country creditor located in.
	Drafts At	Char(6)	lc_head.drafts_at	Specifies the terms of the drafts to be drawn under the LC.
	Drafts At Description	Char(40)	desc is retrieved through a decode	Description of the terms of the drafts to be drawn under the LC.
	Drawee	Char(10)	lc_head.paying_bank	Identifies drawee of drafts to be drawn under LC (paying bank)

Record Name	Field Name	Field Type	Default Value	Description
	Drawee Name	Char(40)	partner.partner_desc	The description from the partner table where partner_id = paying_bank and partner_type = 'BK'.
	Drawee Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Drawee Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Drawee Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Drawee City	Char(20)	addr.city	City bank located in
	Drawee State	Char(3)	addr.state	State, if applicable, where bank located in.
	Drawee Post Code	Char(10)	addr.post	Post code, if applicable, where bank located in.
	Drawee Country	Char(3)	addr.country_id	Country bank located in.
	Negotiating Bank	Char(10)	lc_head.negotiating_bank	Identifies the negotiating bank.
	Negotiating Bank Name	Char(40)	partner.partner_desc	The description from the partner table where partner_id = negotiating_bank and partner_type = 'BK'.
	Negotiating Bank Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Negotiating Bank Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Negotiating Bank Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Negotiating Bank City	Char(20)	addr.city	City bank located in

Record Name	Field Name	Field Type	Default Value	Description
	Negotiating Bank State	Char(3)	addr.state	State, if applicable, where bank located in.
	Negotiating Bank Post Code	Char(10)	addr.post	Post code, if applicable, where bank located in.
	Negotiating Bank Country	Char(3)	addr.country_id	Country bank located in.
	Confirming Bank	Char(10)	lc_head.confirming_bank	Identifies the confirming bank.
	Confirming Bank Name	Char(40)	partner.partner_desc	The description from the partner table where partner_id = confirming_bank and partner_type = 'BK'.
	Confirming Bank Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Confirming Bank Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Confirming Bank Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Confirming Bank City	Char(20)	addr.city	City bank located in
	Confirming Bank State	Char(3)	addr.state	State, if applicable, where bank located in.
	Confirming Bank Post Code	Char(10)	addr.post	Post code, if applicable, where bank located in.
	Confirming Bank Country	Char(3)	addr.country_id	Country bank located in.
	Transferring Bank	Char(10)	lc_head.transferring_bank	Identifies the transferring bank.
	Transferring Bank Name	Char(40)	partner.partner_desc	The description from the partner table where partner_id = transferring_bank and partner_type = 'BK'.

Record Name	Field Name	Field Type	Default Value	Description
	Transferring Bank Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Transferring Bank Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Transferring Bank Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Transferring Bank City	Char(20)	addr.city	City bank located in
	Transferring Bank State	Char(3)	addr.state	State, if applicable, where bank located in.
	Transferring Bank Post Code	Char(10)	addr.post	Post code, if applicable, where bank located in.
	Transferring Bank Country	Char(3)	addr.country_id	Country bank located in.
	Partial Shipment Indicator	Char(1)	lc_head.partial_ship_ind	Indicates whether goods covered by LC can be partially shipped or not.
	Transshipment Indicator	Char(1)	lc_head.transshipment_ind	Indicates whether goods can be transferred to another vessel midway through the voyage.
	Fob Title Pass	Char(6)	lc_head.fob_title_pass	Indicates where the title for goods is passed from the vendor to the purchaser.
	Fob Title Pass Decode	Char(40)	desc is retrieved through a decode	Decode of where the title for goods is passed from the vendor to the purchaser.
	Fob Title Pass Description	Char(45)	lc_head.ob_title_pass_desc	Describes the FOB_TITLE_PASS – could be city name etc.

Record Name	Field Name	Field Type	Default Value	Description
	Transportation to	Char(5)	lc_head.transportation_to	Transportation to location
	transportation to description	Char(40)	outloc.outloc_desc	Description of transportation to lo
	With Recourse Indicator	Char(1)	lc_head.with_recourse_ind	Indicates conditional payment on the part of the bank as instructed by the buyer.
	Latest Shipment Date	Char(14)	lc_head.latest_ship_date	Latest ship date for all Pos included in the LC, formatted to 'YYYYMMDDHH24MISS'
	Earliest Shipment Date	Char(14)	lc_head.earliest_ship_date	Earliest ship date for all Pos included in the LC, formatted to 'YYYYMMDDHH24MISS'
	Letter of Credit Negotiation Days	Number(3) replaces x in the string "DOCUMENTS TO BE PRESENTED WITHIN x DAYS AFTER ISSUANCE OF THE SHIPPING DOCUMENTS BUT WITHIN THE VALIDITY OF THIS CREDIT"	lc.head.lc_neg_days	The number of days to negotiate documents
	Bank's LC reference id	Number(8)	lc_head.bank_lc_id	Bank's LC ref id

Record Name	Field Name	Field Type	Default Value	Description
File Detail	File Type Record Descriptor	Char(5)	THDCM	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	ID of current line being created for output file
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Header Level Comments	Char(2000)	lc_head.comments	Holds any comments that the user has added to the Letter of Credit.
File Detail	File Type Record Descriptor	Char(5)	TDOCS	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	ID of current line being created for output file
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Swift Tag	Char(6)	doc.swift_tag	Identifies individual document types that can be associated with an LC.
	Document ID	Number(6)	req_doc.doc_id	Uniquely identifies the individual documents associated with an LC.
	Body Text	Char(2000)	req_doc.doc_text	Documents associated with a given LC. Description of Goods and Services OR Documents Required OR Additional Conditions OR Narrative
File Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type

Record Name	Field Name	Field Type	Default Value	Description
	File Line Sequence Number	Number(10)	line number in file	ID of current line being created for output file
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Order Number	Number(8)	lc_detail.order_no	PO associated with the LC.
	Item	Number(8)	lc_detail.item	Item on the PO – item is rolled up to the style level, if possible.
	Cost	Number (20,4)	lc_detail.cost	If form_type = 'S'hort then cost is the total cost of the order; if the form_type = 'L'ong then the cost is the unit cost of the item.
	Quantity	Number (12,4)	lc_detail.qty	Total qty of the item for the order on the LC.
	Standard UOM	Char(4)	Rag_style.standard_uom	Standard unit of measure of the quantity of the item for the order on the LC.
	Earliest Ship Date	Char(14)	lc_detail.earliest_ship_date	The earliest date an order on the LC can be shipped, formatted to 'YYYYMMDDHH24MISS'
	Latest Ship Date	Char(14)	lc_detail.latest_ship_date	The latest date an order on the LC can be shipped, formatted to 'YYYYMMDDHH24MISS'
	item description	Char(62)	Desc_look.desc_up	Item's description

Record Name	Field Name	Field Type	Default Value	Description
File Detail	File Type Record Descriptor	Char(5)	TMERC	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	ID of current line being created for output file
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Merchandise Description	Char(2000)	lc_detail.merch_desc	Contains the merchandise description of the field.
File Detail	File Type Record Descriptor	Char(5)	TDTCM	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	ID of current line being created for output file
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Detail Level Comments	Char(2000)	lc_detail.comments	Holds any comments that the user has added to the Letter of Credit detail record.
File Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	ID of current line being created for output file
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Transaction detail line count	Number(10)	ID of current line being created for output file	Sum of the detail lines within a transaction
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type

Record Name	Field Name	Field Type	Default Value	Description
	File Line Identifier	Number(10)	Sequential number Created by program.	ID of current line being created for output file.
	File Record Counter	Number(10)		Number of records/transactions processed in current file (only records between head & tail)

Technical Issues

N/A

[lclrbld.pc]

Design Overview

The Location List Rebuild program performs the rebuilding of all location lists that have static_ind = 'N' and batch_rebuild_ind = 'Y'. It calls package function LOCLIST_BUILD_SQL.REBUILD_LIST to generate LOC_LIST_DETAIL records based on the store and warehouse criteria on the LOC_LIST_CRITERIA table.

Scheduling Constraints

Processing Cycle: Phase II or III

Threading Scheme: LOC_LIST

Restart Recovery

The Logical Unit of Work for this program is location list number. Since this unit of work is unique, there is no need to worry about things like reallocating memory or staggered posting.

The following driving cursor is used (with restart/recovery):

```
select l.loc_list
from loc_list_head l,
     v_restart_loc_list rv,
where rv.driver_value = l.loc_list
     and l.static_ind = 'N'
     and l.batch_rebuild_ind = 'Y'
     and rv.driver_name = :os_restart_driver_name
     and rv.thread_val = :oi_restart_thread_val
     and rv.num_threads = :oi_restart_num_threads
     and l.loc_list > NVL(:oi_restart_loc_list, -999)
order by l.loc_list;
```

Shared Modules

LOCLIST_BUILD_SQL.REBUILD_LIST – for rebuilding the LOC_LIST_DETAIL records for a given location list number.

Function Level Description

main() – Follows Retek standards for the main() function. Logs on to the database, calls init(), process(), and final().

init() – Calls restart_init() to start up restart recovery and bring back the bookmark string in case of a restart.

process() – The controller for most of the processing in the program. This function opens and fetches from the driving cursor according to Retek standards, using array processing. For each record fetched from the driving cursor, rebuild_loc_list() will be called to rebuild location list for the given location list number. When the fetched array has been fully processed, restart_commit() should be called to commit the work, and update the restart/recovery variables.

rebuild_loc_list() – Calls the package function LOCLIST_BUILD_SQL.REBUILD_LIST to rebuild LOC_LIST_DETAIL records for a given location list number.

final() – Calls restart_close() to wrap up restart recovery.

Letter of Credit Amendment Download [lcmdnld]

Design Overview

This purpose of this new Letter of Credit amendment download program, which runs nightly, is to process LC amendments based on the amendments flagged during the day by the Select LC Amendments action on the LC Find form (lcfind.fmb) and then written to the staging table LC_DOWNLOAD. This program should be included in the standard download phase of the batch cycle. The amendments associated to the LCs flagged for a given day will be downloaded in full to a standard format Retek flat file. A separate program will later separate the amendments, sorting by Issuing Bank, into separate SWIFT formatted files which will then be sent to the individual banks (S.W.I.F.T. formats are used when transmitting financial information and are a globally recognized sets of standards). After this program has processed all of the LCs amendments on LC_DOWNLOAD, these records will be deleted and their status on LC_AMENDMENTS will be updated to 'D' for Download.

Affected Tables

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
LC_AMENDMENTS	Yes	Yes	No	Yes	No
LC_HEAD	Yes	Yes	No	No	No
LC_DOWNLOAD	No	Yes	No	No	Yes
ADDR	Yes	Yes	No	No	No
PARTNER	Yes	Yes	No	No	No
SUPS	Yes	Yes	No	No	No
CODE_DETAIL	Yes	Yes	No	No	No
RAG_SKUS	Yes	Yes	No	No	No
WIN_SKUS	Yes	Yes	No	No	No
DOC	No	Yes	No	No	No
REQ_DOC	Yes	Yes	No	No	No

Scheduling Constraints

Processing Cycle:	PHASE 4 (daily)
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	lcmt707 perl script
Threading Scheme:	No threading due to low volume.

Restart Recovery

Restart/recovery for this program is set up at the lc_ref_id level.

The recommended commit counter setting is 1000 records (subject to change based on experimentation).

Program Flow



Shared Modules

LANGUAGE_SQL.GET_CODE_DESC – given the code and code type, this function returns the code description.

LC_AMEND_SQL.AMEND_TEXT—given the order_no, item, amended_field, amended_Desc, original_value, new_value, effect, currency_code, indicator (N if it's a sku, Y if it's a style), this function returns the text field.

LC_AMEND_SQL.STYLE_ROLLUP – given the lc_ref_id, order_no, item, amended_field, amend_no, this function returns the new_value, original_value, effect.

Function Level Description

Init():

This function performs preliminary processing and populates global variables. It will call retek_init to start up restart-recovery and bring back the bookmark string in case of a restart. It will also populate vdate, to be used when writing to the output file. Then call write_fhead to open the output file and write the FHEAD information.

Process():

Call the init_arrays function to initialize the arrays.

This function drives the processing of the program. The driving cursor is fetched here which retrieves all the records from lc_download and lc_head where the lc_download.lc_ref_lc_id value is equal to the lc_head.lc_ref_id value and greater than zero and lc_download.lc_download_type='M'.

This cursor essentially retrieves the bulk of the information that will be written in the header line.

```
EXEC SQL DECLARE c_lc_download CURSOR FOR
```

```
    SELECT ROWIDTOCHAR(l.rowid),  
           h.issuing_bank,  
           h.lc_ref_id,  
           NVL(h.bank_lc_id, ''),  
           h.currency_code,  
           NVL(TO_CHAR(h.confirmed_date,  
                        'YYYYMMDDHH24MISS'), ''),  
           h.beneficiary  
    FROM lc_head h,  
         lc_download l  
   WHERE h.lc_ref_id = l.lc_ref_id  
         AND l.lc_ref_id > nvl(:ps_restart_lc,0)  
         AND l.lc_download_type = 'M'  
   ORDER BY h.issuing_bank, h.lc_ref_id;
```

For each record that is fetched, call `fill_delete_arrays`, which copies the rowid value into an array. Call `write_thead`, which collects the remaining header information and writes it to the output file. Call `get_staple`, which gets the detail information from any staple items. Call `get_style`, which gets the detail information for any item that is a style. Call `write_ttail`, which writes the trailer information to the output file.

Call `update_lc_amendments` to update all the statuses on `lc_amendments` table and call `delete_lc_amendments` to delete all the records on `lc_download` table.

Restart/Recovery and force commit is performed here.

`Final()`:

This function will call `write_ftail` first to write the FTAIL information, then call `retek_close` to wrap up restart-recovery and populate the final error message, if necessary.

`Write_fhead()`:

This function will write the FHEAD line of the output file.

`Write_thead()`:

This function, which takes in the `header_struct`, will write the THEAD line of the output file according to the layout presented later in this document. Most of the variables can be written to the file as is, but some of them will have to be used to fetch additional information from the database. The header structure should then be passed into the `get_all_addresses` function so that all the name and address information can be fetched back for the issuing_bank and the beneficiary. Also, the function `get_current_amount` should be called to figure out the amount of the Letter of Credit only including already fully processed ('D'ownloaded) amendments.

`Write_staple()`:

This function, which takes in the `detail_struct`, will write the TDETL line of the output file according to the layout presented later in this document when the item is a sku. In order to get some of the values, `check_amended_field` will need to be called to get the code description.

`Write_style()`:

This function, which takes in the `detail_struct`, will write the TDETL line of the output file according to the layout presented later in this document when the item is a style. In order to get some of the values, `check_amended_field` will need to be called to get the code description.

`Write_ttext()`:

This function, which takes in the order_no, item, amended_field, amended_desc, original_value, new_value, effect, currency_code and indicator (which is Y if the item is a style and N if the item is a sku), will write a TTEXT record to the output file using the O_amend_text variable passed out from the LC_AMEND_SQL.AMEND_TEXT function call.

Write_ttail():

This function will write the TTAIL line of the output file.

Write_ftail():

This function will write the FTAIL line of the output file.

Create_format_string():

This function formats the strings for the FHEAD, THEAD, TDETL, TTEXT, TTAIL and FTAIL records in the output file.

Init_arrays():

This function initializes the size of the array used for the driving cursor fetch the size of the restart max counter on restart_control.

Get_current_amount():

This function will take in the lc_ref_id and pass out the current amount of the LC. To get the current amount, find the amount from LC_HEAD (which is the original amount that the LC was set for) and then, by joining with the LC_AMENDMENTS table, add to that all of the EFFECT columns for that LC number and whose statuses are in 'D'ownloaded. Note: normally, the net amount is figured out using all of the amendments as long as they have been generated. In this case, because the bank does not have knowledge of these amendments, we want to figure out the current total of the LC using only the amendments they have knowledge of (i.e. ones that have already been downloaded and sent to them). Call compare_value to see if the amount is greater than 9999999999.9999. If it is, skip to next letter of credit and this transaction is not written to the output file.

Compare_value():

Checks to see if the value is greater than or equal to 9999999999999999.

Get_all_addresses():

This function will call get_address, taking in Issuing_bank, Beneficiary, and will fetch back name and address information.

For Bank and Applicant, PARTNER.PARTNER_DESC should be fetched back where the partner_type = 'BK' and the partner_id = the bank (for the banks or the credit_avail_with) or where the partner_type = 'AP' and the partner_id = applicant (for the applicant). Then all the address information (add_1, add_2, add_3, city, state, post, country_id) should be fetched off the addr table. Again, fetch this information using the module = 'PTNR', key_value_1 = 'BK' (for the banks or credit_avail_with) or 'AP' (for the Applicant), and key_value_2 = issuing_bank ID or the applicant ID.

For the Beneficiary, first the supplier name should be fetched back from the SUPS table where Beneficiary = Supplier. Then, off the ADDR table the address information (add_1, add_2, add_3, city, state, post, country_id) should be fetched where the module = 'SUPP' and key_value_1 = beneficiary ID (key_value_2 is NULL).

Get_address():

This function will be called by get_all_addresses and will return the information of name and address.

Check_amended_field():

This function should take in the amended_value and the new_value and then perform a series of if statements that deal with all the possible types of amended fields. If the amended_value is 'PE' (place of expiry), or 'PRT' (purchase type), then the new_value should be passed into get_desc, along with the appropriate code_type. Get_desc will pass back the code_description which should be passed out of the function.

Place_of_expiry LCPE

Purchase_type LCPT

Get_desc():

This function takes in a code and a code_type and fetches the description for that code using LANGUAGE_SQL.GET_CODE_DESC and passes the returned O_code_desc value out of the function.

Update_lc_amendments():

This function will updates the status of the amendments to 'D'ownloaded once they have been successfully processed. This is by rowid. A second check was added because some of the amendments are "rolled up" into one – and so not all records were being updated. Now a check is done after the initial update and if it missed any – it will update them.

Delete_lc_download():

This function will deletes all records from LC_DOWNLOAD that have been fully processed. Done by rowid.

Get_staple():

With the given header information, all of the corresponding detail records for any skus will be retrieved with a cursor that joins lc_amendments and win_skus where lc_amendments.lc_ref_no = lc_ref_no from header information and lc_amendments.item = win_skus.sku or the lc_amendments.item is null and win_skus.sku = (select min(sku) from win_skus), and lc_amendments.status is 'A' and the lc_amendments.amend_no is not null. For each record, call fill_update_arrays to put that record in the update array. Call compare_value to make sure the effect is valid, and call write_staple to write the TDETL line to the output file. Call write_ttext to write the TTEXT line to the output file.

Get_style:

With the given header information, all of the corresponding detail records for any styles will be retrieved with a cursor that joins lc_amendments and rag_skus. For each record, call fill_update_arrays to put that record in the update array. Call style_rollup to retrieve more detail information. Call compare_value to make sure the effect is valid, and call write_staple to write the TDETL line to the output file. Call write_ttext to write the TTEXT line to the output file.

Style_rollup:

Calls the LC_AMEND_SQL.STYLE_ROLLUP package to retrieve the new_value, original_value and effect for a given lc_ref_id, order_no, item, amended_field and amend_no.

Fill_update_arrays:

Adds a record to the array. If more space needs to be allocated for the array, it will do that as well.

Fill_delete_arrays:

Adds a record to the array. If more space needs to be allocated for the array, it will do that as well.

I/O Specification

Output file

It should be noted that for every TDETL line, there will be a TTEXT line immediately following it and containing a text summary of the individual amendment to follow.

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	Keeps track of the record's position in the file by line number
	File Type Definition	Char(4)	LCAM	Identifies file as 'Letter of Credit Amendment'
	File Create Date	Char(14)	create date	current date, formatted to 'YYYYMMDDHH24MISS'
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	Keeps track of the record's position in the file by line number
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Issuing Bank	Char(10)	lc_head.issuing_bank	Used to sort the LCs into individualized bank SWIFT formatted files (using another program) – bank where LC application is headed.
	Issuing Bank Name	Char(40)	partner.partner_desc	The description from the partner table where partner_id = issuing_bank and partner_type = 'BK'.
	Issuing Bank Address 1	Char(30)	addr.add_1	Mandatory line of address.

Record Name	Field Name	Field Type	Default Value	Description
	Issuing Bank Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Issuing Bank Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Issuing Bank City	Char(20)	addr.city	City bank located in
	Issuing Bank State	Char(3)	addr.state	State, if applicable, where bank located in.
	Issuing Bank Post Code	Char(10)	addr.post	Post code, if applicable, where bank located in.
	Issuing Bank Country	Char(3)	addr.country_id	Country bank located in.
	Letter of Credit	Number(8)	lc_detail.lc_ref_id	The LC_REF_ID off the LC_DETAIL table.
	Bank Letter of Credit ID	Char(16)	lc_head.bank_lc_id	The BANK_LC_ID off the LC_HEAD table.
	Currency Code	Char(3)	lc_head.currency_code	The CURRENCY_CODE off the LC_HEAD table.
	Date of Issue/ Transfer of the Credit	Char(14)	lc_head.confirmed_date	The date that the Issuing Bank thinks is the date of issue – when it was officially confirmed, formatted to ‘YYYYMMDDHH24MISS’.
	Current Amount of LC	Number (20,4)		This amount will be calculated in the get_current_amount() function and will be the net amount of the LC calculated only using amendments that have been downloaded. Normally, the net amount is calculated using amendments in the ‘D’ownloaded status.
	Beneficiary	Number(10)	lc.head.beneficiary	Party in favor of which the LC is being issued.

Record Name	Field Name	Field Type	Default Value	Description
	Beneficiary Name	Char(32)	supsup_name	Beneficiary (supplier) name from the SUPS table.
	Beneficiary Address 1	Char(30)	addr.add_1	Mandatory line of address.
	Beneficiary Address 2	Char(30)	addr.add_2	Non-mandatory line of address (can be null).
	Beneficiary Address 3	Char(30)	addr.add_3	Non-mandatory line of address (can be null).
	Beneficiary City	Char(20)	addr.city	City beneficiary located in
	Beneficiary State	Char(3)	addr.state	State, if applicable, where beneficiary located in.
	Beneficiary Post Code	Char(10)	addr.post	Post code, if applicable, where beneficiary located in.
	Beneficiary Country	Char(3)	addr.country_id	Country beneficiary located in.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	Keeps track of the record's position in the file by line number
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check
	Amendment Number	Number(8)	lc_amendments.amend_no	Holds the amendment number for the amendment.
	Order_no	Number(8)	lc_amendments.order_no	Order_no, if applicable, that is attached to the LC that is being amended.
	Item	Number(8)	lc_amendments.item	Item being amended, either a Style or Staple sku.

Record Name	Field Name	Field Type	Default Value	Description
	Value Being Amended	Char(6)	lc_amendments.amended_value	LC Field being amended. Can be any of the following code_types: CODE CODE_DESC
	Value Being Amended Description	Char(40)	code_detail.code_desc	The Value Being Amended decoded (see the above list). Will possibly be used when printing to the SWIFT file MT 707 for clarity.

Record Name	Field Name	Field Type	Default Value	Description
	Original Value of Amended Field	Char(45)	lc_amendments.original_value	Current value of field that is being amended.
	New Value of Amended Field	Char(45)	lc_amendments.new_value	New value of the field that is being amended.
	Description of New Value	Char(40)	code_detail.code_desc	The new value decoded (or fetched from a table, as in the origin_country case)– only applicable to the following amended values: place of expiry, title_pass_location, origin_country, presentation terms, purchase type.
	Sign	Char(1)		If the effect is negative it will be “-“ if the effect is positive it will be “+“.
	Effect	Number (20,4)	lc.amendments.effect	Effect that amendment will have on LC if amendment to change qty or cost of a PO or amount of LC itself.
	Date of Amendment	Char(14)	Lc_amendments.accept_date	Date on which Issuing Bank (or issuing party, in this case the retailer) considers the credit as being amended, formatted to ‘YYYYMMDDHH24MISS’.
Transaction Text	File Type Record Descriptor	Char(5)	TTEXT	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	Keeps track of the record’s position in the file by line number
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check

Record Name	Field Name	Field Type	Default Value	Description
	Amendment Text	Char(2000) {NOTE – size uncertain, within the package, variable allow for char 2000}	text description	A text description of the individual amendment (for each TDETL line of the output file) built by the package LC_AMEND_SQL.AMEND_TEXT.
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	Keeps track of the record's position in the file by line number
	Control Number File Line Count	Number(10)	total detail lines	sum of all transaction lines, not including file header and trailer

Technical Issues

N/A

[lcmt700.perl]

Design Overview

The S.W.I.F.T file is in a different file layout than the standard Retek interface file format. This program will convert the standard Retek flat file into the bank specific S.W.I.F.T. MT 700 output files. One output file will be created for each issuing bank in the interface file. A Perl script program will do the file conversion.

Scheduling Constraints

This program should be run after Letter of Credit application download program (lcadnld.pc).

Function Level Description

main program flow:

check for the correct number of arguments (should be 2: inputfile outputfile)

open the input file

read the first line of the file and verify that it's an FHEAD line – if not, exit.

Initialize counter variables. These will be set throughout the program and then totaled.

while a line is read from the input file:

if line begins with THEAD:

call populate_THEAD()

set ctrl to 1

if the issuing bank has changed from the previous issuing bank, then:

open a new file. Filename should be outputfile_issuing_bank (e.g. output1_NORWEST)

call init_variables()

elsif line begins with TDOCS:

call populate_TDOCS()

elsif line begins with TDETL:

```
call populate_TDETL()

elsif line begins with TMERC

read in the merch_desc from the input file.

elsif line begins with THDCM or TDTCM

skip to next line

elsif line begins with TTAIL:

if this is not the first MT700 for this bank:

print the ASCII code 3 separator

total the counters.

call write_MT700()

call write_MT701_45B()

call write_MT701_46B()

call write_MT701_47B()

elsif line begins with FTAIL

if at least one bank has been processed, close the output file

stop reading lines from the input file

else

print an error message about an unexpected line type and exit

now that the while loop is done, close the input file

init_variables()

Assign empty values to global variables that are unique to a single MT700/701
application.

populate_THEAD()

Read substrings from the current line into global variables to be used in the
write_* functions. Check the SWIFT file format and the lcadnld file format to
see if any truncation or manipulation must be done.
```

populate_TDOCS()

Read the swift tag and body text from the current line. Replace consecutive spaces in the body text with a single space. The body text should become a part of whatever field the swift tag indicates.

populate_TDETL()

Read the order number, item, cost, quantity, earliest ship date, latest ship date, and description from the current line.

Populate()

if the cost is NULL, set it to '0'

if the quantity is NULL, set it to '0'

create a string as shown in field 45B of the MT701 form below. How this string is built depends on the form type, which is read in populate_THEAD().

push this string into the details array.

write_MT700()

print the line "MT/PRIORITY:700 02" followed by a newline

print all fields as indicated in the file format below in the case of fields 71B, 48, 78 and 72, the tag will be concatenated to the front of each string and a '~+' will be concatenated to the end. Then it will be split into an array and each element of the array will be fed to the function trunc_wrap which will separate each of those strings into 65 char lines. Each line will then be printed to the output file.

write_MT701_45B()

if the MT701 field 45B variable is not null:

print the header line "MT/PRIORITY: 701 02"

print the field 27 line

print the field 20 line

Concatenate 45B to the front of the 45B field string and a '~+' will be concatenated to the end. Then it will be split into an array and each element of the array will be fed to the function trunc_wrap which will separate each of those strings into 65 char lines. Each line will then be printed to the output file.

write_MT701_46B()

if the MT701 field 46B variable is not null:

print the header line “MT/PRIORITY: 701 02”

print the field 27 line

print the field 20 line

Concatenate 46B to the front of the 45B field string and a ‘~+’ will be concatenated to the end. Then it will be split into an array and each element of the array will be fed to the function `trunc_wrap` which will separate each of those strings into 65 char lines. Each line will then be printed to the output file.

`write_MT701_47B`

if the MT701 field 47B variable is not null:

print the header line “MT/PRIORITY: 701 02”

print the field 27 line

print the field 20 line

Concatenate 47B to the front of the 45B field string and a ‘~+’ will be concatenated to the end. Then it will be split into an array and each element of the array will be fed to the function `trunc_wrap` which will separate each of those strings into 65 char lines. Each line will then be printed to the output file.

`trunc_wrap()`

arguments: 1) input string

maximum number of characters in a line

maximum number of lines

declare local variables to hold the total number of lines, the current line, and an array to store all lines

if the length of the input string is evenly divisible by the max number of chars in a line

total number of lines = `int(length of input string / max number of chars in a line)`

else

total number of lines = `int(length of input string / max number of chars in a line)`
+ 1

for each line (but not more than the max number of lines)

create a string from the next (max chars in line) number of characters from the input string

if there are less than (max chars in line) remaining in the input string, right pad current line with spaces until length = max chars in line

push the current line into the storage array

return the storage array

I/O Specification

The SWIFT MT 700 output file should be in the following format:

Most output fields are contained in their own line (or 3-4 line for addresses).

Each application consists of four parts, one MT 700 and three MT 701s, which are ordered through the Sequence of Total field: ex. ':27:1/4 MT 700' is the first (MT 700) part of the application

MT 700 and MT 701s will be mingled in the same file.

Each record starts with a colon and a SWIFT field identifier, followed by another colon: ex. ':40A:?'-

Each application is separated by a line with only the ASCII 3 symbol (a heart) on it.

Examples of how individual lines of the MT 700 or MT 701 should look:

:27:1/4

:40A:IRREVOCABLE

:20:29893098

:23:NOREF

:31C:910906

:31D:911022DALLAS

:51D:NORTHERN TRUST INT'L BANKING CORP.

ONE WORLD TRADE CENTER

SUITE 3941

NY, NY 10048 USA

The layout of the S.W.I.F.T MT 700 (Issue of a Documentary Credit) file is as follows:

SWIFT I.D. DATA TYPE CODES (refer to SWIFT User Handbook – Standards General Information – October 1998 release for formatting information):

SWIFT I.D.	Data Type	Description	What needs to go in the SWIFT field: Retek Standard Download Format Field(s) and New Necessary Sizes and How-To-Arrange-It.	Comments
27 – Sequence of Total	1n/1n	Number of parts to each application (1 700 and 3 701s are the max so n/4)	'1/4'	Will be 1/4 for LC application body.
40A – Form of Documentary Credit	24x	Defines condition for guarantee of funds and acceptance of LC credit. Can be REVOCABLE or IRREVOCABLE. Then, if applicable, followed by a blank character and the word TRANSFERABLE.	THEAD: - Form of Letter of Credit–I, Char(11) ' ' - Form of Letter of Credit–II, Char(12)	If the revocable_ind (Form of LC – I) is 'Y' then REVOCABLE, if 'N' then IRREVOCABLE. If the transferable_ind (Form of LC – II) is 'Y', then TRANSFERABLE, otherwise nothing.
20 – Documentary Credit Number	16x	LC number. The one assigned by the Sender (retailer).	THEAD - Letter of Credit ID, Number(8) OR - Bank LC Ref ID, Char(16) – if available	Should the Bank LC Ref ID be used if available? Current position – For pre-issued Bank LC Ref IDs, if the bank LC ref ID is populated for the LC, this should be passed to the output file. If it is not available, in internally generated LC ref ID should be passed.

SWIFT I.D.	Data Type	Description	What needs to go in the SWIFT field: Retek Standard Download Format Field(s) and New Necessary Sizes and How-To-Arrange-It.	Comments
31C – Date of Issue	6!n	Date on which issuing bank considers the LC as being issued. Blank = date message sent.	NULL	This could be one of three possibilities. The application date or the download date or NULL (which would imply the download date). Current position – use NULL which will be taken as the download date by the receiver.
31D – Date and Place of Expiry	6!n29x	Place and latest date that documents may be presented.	THEAD - Expiration Date, Date - Place of Expiry Description, Char(29)	Put date in correct format (YYMMDD) and concatenate with truncated description (truncate from 40 to 29).
50 – Applicant	4*35x	Party and address on whose behalf the LC is being issued.	THEAD - Applicant Name, Char(35) nl - Applicant Address I, Char(30) nl - Applicant Address II, Char(30) nl - Applicant City, Char(16) ', ' - Applicant State, Char(3) ' ' - Applicant Post Code, Char(10) ' ' - Applicant Country, Char(3)	Truncate applicant name to 35. New lines after name, address I and address II. Truncate city to 16 from 20 to accommodate city, state, postal code and country in line 4.

SWIFT I.D.	Data Type	Description	What needs to go in the SWIFT field: Retek Standard Download Format Field(s) and New Necessary Sizes and How-To-Arrange-It.	Comments
51D – Issuing Bank	Option D- [/1!a][34x] 4*35x	This field specifies the name and address of the financial institution which is acting on behalf of itself, or a customer, to initiate the transaction.	THEAD - Issuing Name, Char(35) nl - Issuing Address I, Char(30) nl - Issuing Address II, Char(30) nl - Issuing City, Char(16) ', ' - Issuing State, Char(3) ' ' - Issuing Post Code, Char(10) ' ' - Issuing Country, Char(3)	Truncate the issuing bank name to 35. New lines after name, address I and address II. Truncate city to 16 to accommodate city, state, postal code and country in line 4.
59 – Beneficiary	[/34x] 4*35x	Specifies the party and address in favor of which the LC is being issued.	THEAD - Beneficiary Name, Char(35) nl - Beneficiary Address I, Char(30) nl - Beneficiary Address II, Char(30) nl - Beneficiary City, Char(15) ', ' - Beneficiary State, Char(3) ' ' - Beneficiary Post Code, Char(10) ' ' - Beneficiary Country, Char(3)	Since an account can not be defined within RTM for the beneficiary, the optional [/34x] Account will not be used. Truncate beneficiary name to 35. New lines after name, address I and address II. Truncate city to 16 from 20 to accommodate city, state, postal code and country in line 4.

SWIFT I.D.	Data Type	Description	What needs to go in the SWIFT field: Retek Standard Download Format Field(s) and New Necessary Sizes and How-To-Arrange-It.	Comments
32B – Currency Code, Amount	3!a15d	Currency Code and amount of LC.	THEAD - Currency Code, Char(3) - Amount, Number(20,4)	Truncate Amount to the rightmost 14 characters. Insert a comma where the implied decimal point would be. The resulting string will have 15 characters including the comma. Current position - given that RTM currency amounts can be up to a (20,4), any LC applications with amounts not able to fit into a 15d field will need to be rejected and not downloaded.
39A – Percentage Credit Amount Tolerance	2n/2n	Tolerance relative to LC credit amount as a percentage plus and/or minus that amount. First field spec. positive tol, 2nd negative tol.	THEAD If Amount_type = 'A' pproximately then Variance Percent, Char(2) '/' Variance Percent, Char(2) If Amount_type = 'E' xact then - this field is NULL and field 39B is used.	Currently in RTM, can only enter one variance, therefore positive and negative variance percents are always the same. If this field contains a value, 39B cannot, and vice versa.

SWIFT I.D.	Data Type	Description	What needs to go in the SWIFT field: Retek Standard Download Format Field(s) and New Necessary Sizes and How-To-Arrange-It.	Comments
39B – Maximum Credit Amount	13x	Further qualifies LC credit amount by using one of following terms: UP TO, MAXIMUM, NOT EXCEEDING	THEAD If Amount_type = 'E'xact or - Specification = 'M'aximum then 'MAXIMUM'	RTM currently does not have fields that map directly to this. RTM currently has Amount Type and Specification. Amount Type can be Exact or Approximately. Specification can be 'M'aximum or others like 'Insurance to be Paid' Additionally. Current Position – if Specification = Maximum, set this field to MAXIMUM or if Amount Type = 'Exact, set this field to MAXIMUM. If this field contains a value, 39B cannot, and vice versa.

SWIFT I.D.	Data Type	Description	What needs to go in the SWIFT field: Retek Standard Download Format Field(s) and New Necessary Sizes and How-To-Arrange-It.	Comments
41a – Available With. . . By . . .	Option A – 4!a2!a2!c[3!c] 14x Option D – 4*35x 14x	Identifies bank authorized to pay, accept or negotiate credit. Followed by how credit is available: BY PAYMENT BY ACCEPTANCE BY NEGOTIATION BY DEF PAYMENT BY MIXED PYMT If either last two then fields 42P and 42 M necessary (one or the other).	THEAD - Credit With Name, Char(35) nl - Credit With Address I, Char(30) nl - Credit With Address II, Char(30) nl - Credit With City, Char(16) , ‘ - Credit With State, Char(3) ‘ - Credit With Post Code, Char(10) ‘ - Credit With Country, Char(3) nl - Presentation Terms Description, Char(14)	Since RTM has no facilities to maintain BICs, option D will always be used for this field (i.e. 41D) Truncate credit available with name to 35. New lines after name, address I and address II. Truncate city to 16 from 20 to accommodate city, state, postal code and country in line 4. Use the decoded presentation terms value to arrive at one of these ‘by’ options: BY PAYMENT BY ACCEPTANCE BY NEGOTIATION BY DEFPAYMENT BY MIXED PYMT
42C – Drafts at . . .	3*35x	Specifies the terms of the drafts to be drawn under the LC.	THEAD - Drafts At Description, Char(40)	Since the decode can be up to 40 chars, line wrap by word to the next line if required.

SWIFT I.D.	Data Type	Description	What needs to go in the SWIFT field: Retek Standard Download Format Field(s) and New Necessary Sizes and How-To-Arrange-It.	Comments
42D – Drawee	Option D – [/1!a][[/34x] 4*35x	Identifies drawee of drafts to be drawn under the LC. Optional act # line must not be present	THEAD - Paying Bank Name, Char(35) nl - Paying Bank Address I, Char(30) nl - Paying Bank Address II, Char(30) nl - Paying Bank City, Char(16) ', ' - Paying Bank State, Char(3) ' ' - Paying Bank Post Code, Char(10) ' ' - Paying Bank Country, Char(3)	Truncate the paying bank name to 35. New lines after name, address I and address II. Truncate city to 16 to accommodate city, state, postal code and country in line 4.
43P – Partial Shipments	1*35x	Specifies whether or not partial shipments permitted.	THEAD If - Partial Shipment Indicator = 'Y' then 'ALLOWED' else if - Partial Shipment Indicator = 'N' then 'NOT ALLOWED'	

SWIFT I.D.	Data Type	Description	What needs to go in the SWIFT field: Retek Standard Download Format Field(s) and New Necessary Sizes and How-To-Arrange-It.	Comments
43T – Transshipment	1*35x	Specifies whether or not transshipments permitted.	THEAD If - Transshipment Indicator = ‘Y’ then ‘ALLOWED’ else if - Transshipment Indicator = ‘N’ then ‘NOT ALLOWED’	
44A – Loading on Board /Dispatch/ Taking in Charge at/from ...	1*65x	Specifies place of dispatch or taking in charge of the goods or loading on board.	THEAD - Fob Title Pass Decode, Char(40) ’ ‘ - Fob Title Pass Description, Char(45)	Current position – Will use FOB title pass decode and description. Since the maximum length of both of these fields combined is 85, truncate both appropriately to fit within 65. Take full length of decode text (not including trailing spaces) and concatenate with truncated description.
44B – For Transportation to ...	1*65x	Specifies final destination of goods.	THEAD Transport_to, Char(40)	This is a new column in the input file
44C – Latest Date of Shipment	6!n	Specifies the latest date for loading/ dispatch/ taken in charge.	THEAD - Latest Shipment Date, Date(6)	Use date format YYMMDD.
71B – Charges	6*35x	Charge to be borne by the beneficiary. If blank then all charges except negotiation And transportation. Charges to be borne by applicant.	TDOCS If document type = ‘Charges’ then -Text Body	This is a new document type created for the code type ‘DOCT’.

SWIFT I.D.	Data Type	Description	What needs to go in the SWIFT field: Retek Standard Download Format Field(s) and New Necessary Sizes and How-To-Arrange-It.	Comments
48 – Period for Presentation	4*35x	Number of days for negotiation, payment, and acceptance. Absence = 21 day default where applicable.	THEAD -Narrative - Letter of Credit Negotiation Days, Char(3)	Hardcode this narrative and include the negotiation days: “DOCUMENTS TO BE PRESENTED WITHIN <Negotiation Days> DAYS AFTER DATE OF ISSUANCE OF THE SHIPPING DOCUMENTS BUT WITHIN THE VALIDITY OF THIS CREDIT”.
49 – Confirmation Instructions	7!x	Contains confirmation instructions for the Receiver, as follows: CONFIRM MAY ADD WITHOUT	'WITHOUT'	RTM currently does not have fields that map directly to this. IMC hard-coded 'WITHOUT' Current position – since this is a mandatory field, will default 'WITHOUT'.
78 – Instructions to the Paying/ Accepting/ Negotiating Bank	12*65x	Specifies instructions to the paying, accepting or negotiating bank.	TDOCS If document type = 'Bank Instructions' then - Text Body	New document type being created for code type 'DOCT'.

SWIFT I.D.	Data Type	Description	What needs to go in the SWIFT field: Retek Standard Download Format Field(s) and New Necessary Sizes and How-To-Arrange-It.	Comments
57a – “Advise Through” Bank	Option A – [!a]/[34x] 4!a2!a2!c[3!c] Option B – [!a]/[34x] [35x] Option D – [!a]/[34x] 4*35x	Name of bank, if different from Receiver (i.e. issuing bank) through which the LC is to be routed/ advised/ confirmed to the Beneficiary.	THEAD - Advising Name, Char(35) nl - Advising Address I, Char(30) nl - Advising With Address II, Char(30) nl - Advising With City, Char(16) ‘ ‘ - Advising With State, Char(3) ‘ ‘ - Advising With Post Code, Char(10) ‘ ‘ - Advising With Country, Char(3)	Since RTM has no facilities to maintain BICs or party identifiers, option D will always be used for this field (i.e. 41D) without [!a]/[34x] party identifier. RTM currently has fields to accommodate the advising and confirmation bank. IMC placed both bank in this field if they were different, exceeding the 4*35 length. Current position – will place only the advising bank in this field. Truncate advising bank name to 35. New lines after name, address I and address II. Truncate city to 16 from 20 to accommodate city, state, postal code and country in line 4.
72 – Sender to Receiver Information	6*35x	Specifies additional information to the receiver.	TDOCS If document type = ‘Sender Instructions’ then - Text Body	This is a new document type created for code type ‘DOCT’.

MT701 Mapped to Retek Fields:

SWIFT I.D.	Data Type	Description	What needs to go in the SWIFT field: Retek Standard Download Format Field(s) and New Necessary Sizes and How-To-Arrange-It.	Comments
27 – Sequence of Total	1n/1n	Number of parts to each application (if first MT701 then it would be 2/4, assuming there were three 701s)	‘2/4’ for first record, ‘3/4’ for second record, ‘4/4’ for third record	Will be 2/4, 3/4 and 4/4 for three MT 701s.
20 – Documentary Credit Number	16x	LC number. The one assigned by the Sender (retailer)	THEAD - Letter of Credit ID, Number(8) OR - Bank LC Ref ID, Char(16) – if available	Should the Bank LC Ref ID be used if available? Current position – For pre-issued Bank LC Ref IDs, if the bank LC ref ID is populated for the LC, this should be passed to the output file. If it is not available, in internally generated LC ref ID should be passed.

SWIFT I.D.	Data Type	Description	What needs to go in the SWIFT field: Retek Standard Download Format Field(s) and New Necessary Sizes and How-To-Arrange-It.	Comments
45B – Description of Goods and Services	100*65x	Description of goods and services.	TDOCS If Swift Tag = '45B' then Body Text (truncate and wrap) If Form Type = 'S' (short) then TDETL description, Char(65) nl TDETL Order No., Number(8) ': ' TDETL total cost(qty*cost), Number(32) ' ' TDETL Earliest Ship Date, Date(8) ' ' TDETL Latest Ship Date, Date(8) nl THEAD purchase type decode, Char(40) Elif Form Type = 'L' (long) then TDETL description, Char(65) nl TDETL Order No., Number(8) ': ' TDETL Item, Number(8) ' ' TDETL Cost, Number(32) ' ' TDETL Quantity, Number(12,4) TDETL Earliest Ship Date, Date(8) ' ' TDETL Latest Ship Date, Date(8) nl THEAD purchase type decode, Char(40)	Description field must be truncated to 65 characters. It is a varchar2(2000) field on ordlc. The purchase type field should contain the decoded description. See description below for example. Also refer to the spreadsheet below for a more specific layout.

SWIFT I.D.	Data Type	Description	What needs to go in the SWIFT field: Retek Standard Download Format Field(s) and New Necessary Sizes and How-To-Arrange-It.	Comments
46B – Documents Required	100*65x	Documents required.	TDOCS if - Swift Tag = '46B' then - Doc Text	Current position – Since the MT701 is always used, required documents will always use SWIFT tag 46B. Issue – As well, it appears that one MT701 can be used for each of description of goods and services, documents required and additional conditions. Does this mean that all 3 fields could be used in each MT701 thereby allowing for up to 3*6500 (19500) chars?
47B – Additional Conditions	100*65x	Any further conditions.	TDOCS if document type = 'Additional Instructions' then - Text Body	Current position – Since the MT701 is always used, special instructions (additional conditions) will always use SWIFT tag 47B. Issue – As well, it appears that one MT701 can be used for each of description of goods and services, documents required and additional conditions. Does this mean that all 3 fields could be used in each MT701 thereby allowing for up to 3*6500 (19500) chars?

[LCMT798.perl]

Design Overview

This new perl script will take an input file consisting of SWIFT tags and reformat it into a standard Retek field format. The input file will contain Letter of Credit (L/C) activity data for charges and drawdowns.

Shared Modules

None

Input Specifications

Swift Tag	Description	Required?	Datatype	Retek Field
20 – Transaction Reference Number	The Sender's unambiguous identification of the transaction. Its detailed form and content are at the discretion of the Sender.	Yes	16x – Transaction Reference Number	Bank L/C ID Lc_head.bank_lc_id Varchar2(16)
12 – Type of Financial Instrument	This field classifies the financial instrument by a description or proprietary code.	Yes	Option A- :4!c/[8c]/30x :4!c – Qualifier / - Delimiter [8c] – Issuer Code / - Delimiter 30x - Type	This field will contain a constant identifier – '798'.
77E – Proprietary Message	This field contains the proprietary message in a format agreed to by the Sender and the Receiver.	Yes	Option E- 73x [n*78x]	This field will contain the information below (fields 21, 23, 32C, 32D, 71A, 33A, 72). Carriage return, Line feed, Colon 'CrLf:' will be used to separate fields included in this 77E. For example:

Swift Tag	Description	Required?	Datatype	Retek Field
				:77E:'CrLf' :21:10004321:CrLf' :32C:990121USD1045 etc... There may be multiple 77E's in one file.
21 – Related Reference	This field specifies, in an unambiguous way, a message or transaction identifier which is normally included as part of the information supplied with the message or transaction itself, and can subsequently be used to distinguish the message or transaction identified from other messages or transactions.	No	16x	P/O Number Lc_activity.order_no Number(8)
23 – Further identification	This field specifies the type of transaction being confirmed, as well as the settlement method used.	No	16x	Invoice Number Lc_activity.invoice_no Varchar2(15)
32C – Date and Amount	This field specifies the currency code and amount in a transaction and a corresponding date.	No	Option C- 6!n3!a15d 6!n – Date 3!a – Currency 15d – Amount	Charges Credited (this is interpreted as a positive amount) Date will be in format YYMMDD The integer part of the Amount must contain at least one digit. A decimal comma ',' is mandatory and is included in the maximum length. Lc_activity.amount Number(20,4)

Swift Tag	Description	Required?	Datatype	Retek Field
				Lc_activity.currency_code Varchar2(3) Lc_activity.activity_date Date
32D – Date and Amount	This field specifies the currency code and amount in a transaction and a corresponding date.	No	Option D- 6!n3!a15d 6!n – Date 3!a – Currency 15d – Amount	Charges Debited (this is interpreted as a negative amount) Date will be in format YYMMDD The integer part of the Amount must contain at least one digit. A decimal comma ‘,’ is mandatory and is included in the maximum length. Lc_activity.amount Number(20,4) Lc_activity.currency_code Varchar2(3) Lc_activity.activity_date Date
33A – Date and Amount	This field specifies the currency code and amount in a transaction and a corresponding date.	No	Option A- 6!n3!a15d 6!n – Date 3!a – Currency 15d – Amount	Date, currency, amount of drawing (this is interpreted as a positive amount) Date will be in format YYMMDD The integer part of the Amount must contain at least one digit. A decimal comma ‘,’ is mandatory and is included in the maximum length.

Swift Tag	Description	Required?	Datatype	Retek Field
				Lc_activity.amount Number(20,4) Lc_activity.currency_code Varchar2(3) Lc_activity.activity_date Date
33C – Date and Amount	This field specifies the currency code and amount in a transaction and a corresponding date.	No	Option A- 6!n3!a15d 6!n – Date 3!a – Currency 15d – Amount	Date, currency, amount of drawing (this is interpreted as a negative amount) Date will be in format YYMMDD The integer part of the Amount must contain at least one digit. A decimal comma ‘,’ is mandatory and is included in the maximum length. Lc_activity.amount Number(20,4) Lc_activity.currency_code Varchar2(3) Lc_activity.activity_date Date
72 – Sender to Receiver Information	This field specifies instructions or additional information for the Receiver, Intermediary, Account With Institution or Beneficiary Institution.	No	6*35x	Comments Lc_activity.comment Varchar2(2000)
18A – Number of Repetitive Parts	This field specifies the number of times the repetitive part(s)/sequence(s), directly before or after this field appears in the message.	No	Option A- 5n – Number of Repetitive Parts.	Number of 77E’s contained within the file.

Output Specifications

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)	line number in file	ID of current line being created for output file.
	File Type Definition	Char(4)	LCCH	Identifies file as 'Letter of Credit Changes'
	File Create Date	Char(14)	Create date	current date, formatted to 'YYYYMMDDHH24MISS'
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	ID of current line being created for output file.
	Bank Letter of Credit Reference ID	Char2(16)	SWIFT tag 20	Bank L/C ID
	Order Number	Number(8)	SWIFT tag 21	Contains the order number that is attached to the letter of credit.
	Invoice Number	Char2(15)	SWIFT tag 23	Identifies the Issuing Bank's invoice number to which the drawdown refers. This field does not correspond to a Retek invoice number.
	Transaction Number	Char2(10)	Null	Identifies the amendment number or actual transaction number assigned by the bank.
	Transaction Code	Char2(6)	If the transaction is a Bank Charge – 'B' If the transaction is a Drawdown – 'D'	Identifies the type of transaction that occurred. The type is determined by what detail fields are received for the record. If the record contains a 33A this field will get a 'D'. If the record contains either a 32C or 32D this field will get a 'B'.
	Amount Sign	Char2(1)	SWIFT 33A, 33C	If the record contains a 33A field leave a blank space in this field.

Record Name	Field Name	Field Type	Default Value	Description
			SWIFT 32C, 32D	If the record contains a 33C field this field should contain a ‘-’. If the record contains a 32C field leave a blank space in this field. If the record contains a 32D field this field should contain a ‘-’.
	Amount	Number(20)	SWIFT 33A, 33C SWIFT 32C, 32D	Holds the amount of the activity. This field will have 4 implied decimal places. If SWIFT 32C or 32D (Bank Charge) contains a value, use the amount from this field. If SWIFT 33A or 33C (Drawdown) contains a value, use the amount from this field.
	Currency Code	Char2(3)	SWIFT 33A, SWIFT 32C, 32D	Contains the activity’s currency code. If SWIFT 32C or 32D (Bank Charge) contains a value, use the currency from this field. If SWIFT 33A (Drawdown) contains a value, use the currency from this field.
	Activity Date	Char2(8)	SWIFT 33A, SWIFT 32C, 32D	Holds the date that the activity took place. Formatted to ‘YYYYMMDD’ If SWIFT 32C or 32D (Bank Charge) contains a value, use the date from this field. If SWIFT 33A (Drawdown) contains a value, use the date from this field.
	Comments	Char2(2000)	SWIFT tag 72	Holds any comments for the activity.
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type

Record Name	Field Name	Field Type	Default Value	Description
	File Line Identifier	Number(10)	Sequential number Created by program.	ID of current line being created for output file.
	File Record Counter	Number(10)		This will contain the number of FDETL lines processed.

Function Level Description

Bank Charges and Drawdowns will be received from the bank in a SWIFT lcmt798 file. This perl script will take this lcmt798 file and convert it to a standard Retek file. The file produced by this script will be uploaded into the Retek system by a new upload program (lcup798.pc).

Refer to the file layouts above for formatting and field mapping. A file containing a Bank Charge will be identifiable because it will contain a 32C or 32D field, Drawdown files will not. A field containing a Drawdown will be identifiable because it will contain a 33A or 33C field, Bank Charge files will not.

There can be multiple input files run with this program at one time. The input file will contain at least one 77E. Within the 77E will be the details of the record. A single file may contain multiple 77E's. The 18A field will contain the number of 77E records within the file. Use the value in this field to ensure the correct number of 77E records were received.

The perl script will do the following:

Check number of arguments for program name, outputfile, input file (and possibly more input files)

Create handle for output file

Write FHEAD

Open the input file and read in one line at a time call function process_one_file

Process_one_file

Read in and disregard the header information until the start of the detail information

Call function init_variables

While loop (read in one line at a time)

If the line contains the string PRIORITY:798

Call write_FDETL

Increment line counter

Else if the line contains the string :03:

Go to the next line

Else if the line contains the string :20:

Read in up to 16 characters starting with the 4th (after the :20:) into a variable

Pad to 16 with blank spaces

Else if the line contains the string :12:

Go to the next line

Else if the line contains the string :77E:

Check to see if :21: is contained on this line

If it is, read in up to 12 characters starting with the 4th (after the :21:) into a variable

Pad to 8 with blank spaces

Else if the line contains the string :21:

If it is, read in up to 12 characters starting with the 4th (after the :21:) into a variable

Pad to 8 with blank spaces

Else if the line contains the string :23:

If it is, read in up to 15 characters starting with the 4th (after the :23:) into a variable

Pad to 15 with blank spaces

Else if the line contains the string :32C: or :32D: or :33A: or :33C:

Read the first 5 characters into a variable (code)

Read the next 6 characters into a variable (date)

Read the next 3 characters into a variable (currency code)

Read the next up to 15 characters into a variable (amount)

The amount contains at least one digit and a comma. Replace the comma with a period

Multiply the amount by 10000

Add .5 to the amount and then chop off everything after the decimal point (round the number)

Zeropad the amount to 20 places

Change the dates to have 4 digit years instead of 2 digit years

If the code is :32C:, transaction_code = B, amount sign is a blank space

If the code is :32D:, transaction_code = B, amount sign is '-'

If the code is :33A:, transaction_code = D, amount sign is a blank space

If the code is :33C:, transaction_code = D, amount sign is '-'

Else if the line contains the string :72:

Read in all the comments line and concatenate them together into one variable

Else if the line contains the string :18A:

Read in up to 5 characters starting with the 5th (after the :18A:) into a variable

Pad to 5 with blank spaces

End loop

Function zeropad

Pads comments to 2000 characters

Function inti_variables

Initializes variables

Function write_fDETL

Writes variables to output file.

Scheduling Considerations

This program should be run before lcup798.pc in phase II.

Restart / Recovery

None

[lcup798.pc]

Design Overview

The program will read data from the input file, validate it, and insert it into the lc_activity table. If a record fails validation, it will be written to a reject file (that can be reprocessed by lcup798 after errors have been corrected).

Function Level Description

init

Initialize restart/recovery; open input file and reject file.

Be sure to check if this is a restart or not. If it's a fresh start, check to make sure that the file type (from the FHEAD line) is 'LCCH'; fatal error if it isn't

process

In a loop, read a line from the input file (use the retek_get_record library routine); validate (call validate_record), write to an array for future insert (or write to reject file, if validation failed). Do an array insert of the validated records by calling post_records whenever the commit point is reached. Quit when the FTAIL line is read (validate the FTAIL line to check number of FDETL lines)

validate_record

Check that all dates are valid dates and all numbers are valid numbers. Call validate_lc_ref, validate_order, and validate_currency to validate the bank lc ref id, order number, and currency code against those in the database. Note that it is OK for the order number and/or the invoice number to be blank, but the bank lc ref id, amount, transaction code, activity date, and currency code must be given.

validate_lc_ref:

Check that a given bank lc_id exists on the lc_head table and get the corresponding lc_ref_id for insert into lc_activity.

Only check against lc_head records with a status of 'C' onfirmed.

validate_order:

Check that a given order number exists on the lc_detail table for a given lc_ref_id. If it does not exist on lc_detail, check the lc_amendments table (it is acceptable if an order exists on either of these tables).

validate_currency

Check that given currency code exists on the currencies table. Call `get_exchange_rate` to get the exchange rate for this currency.

`get_exchange_rate`:

Call the `CURRENCY_SQL.GET_RATE` function to get the exchange rate for a given currency code and activity date. Use 'L' as the exchange type (this will pick up the letter of credit exchange rate if it exists and default to consolidation or operational exchange rate otherwise). Pass in the activity date as the effective date.

`post_records`

Array insert records into the `lc_activity` table. All columns on the table should be inserted, but note that some may be NULL.

`LC_REF_ID`—must always be given

`ORDER_NO`—may be null

`INVOICE_NO`—may be null

`TRANS_NO`—may be null

`TRANS_CODE`—must be given

`AMOUNT`—must be given

`ACTIVITY_DATE`—must be given

`CURRENCY_CODE`—must be given

`EXCHANGE_RATE`—must be given

`COMMENTS`—may be null)

(No duplicate checking.).

`final`

Restart/recovery close; close input and reject files.

Input Specifications

Input file:

Record Name	Field Name	Field Type	Default Value	Description
FHEADFile Header	line typeFile Type Record Descriptor	char(5)Char(5)	'FHEAD'FHEAD	marks beginning of fileIdentifies file record type
	line numberFile Line Identifier	number(10)Numeric ID(10)	0000000001Sequential number Created by program.	unique line identifierID of current line being created for output file.
	file type IDFile Type Definition	char(4)Char(4)	'LCCH'PRMD	identifies this as an LC 798 file-Letter of Credit ChargesIdentifies file as 'PROM Download'
	file create dateFile Create Date	char(14)Char(14)	create date	File create date 'YYYYMMDDHH24MISS' formatcurrent date, formatted to 'YYYYMMDDHH24MISS'.
FDETLFile Detail	line typeFile Type Record Descriptor	char(5)Char(5)	'FDETL'FDETL	marks beginning of a transactionIdentifies file record type
	line numberFile Line Identifier	number(10)Numeric ID(10)nu	Sequential number Created by program.	unique line identifierID of current line being created for output file.
	bank letter of credit reference ID	char(16)	SWIFT tag 20	Bank's LC ref ID
	Order number	Number(8)	SWIFT tag 21	order number attached to LC May be blank.

Record Name	Field Name	Field Type	Default Value	Description
	invoice number	char(15)	SWIFT tag 23	Note: this is NOT a RMS invoice number, just a reference invoice number from the issuing bank (to which the drawdown refers) May be blank
	transaction number	char(10)		amendment number or transaction number assigned by bank—can be null
	transaction code	char(6) (why not char(1)?	B or D	‘B’ank charge or ‘D’rawdown
	amount	number(21)	SWIFT tag 33A, 71A	(This is a 20-digit number with a leading – sign or blank and 4 implied decimal places.) Amount of charge or drawdown.
	currency code	char(3)	SWIFT 33A, 71A	Currency that the amount is in.
	activity date	char(8)	SWIFT 33A, 32C, 32D	activity date, ‘YYYYMMDD’ format
	comments	char(2000)	SWIFT tag 72	Any comments associated with activity. Can be null.
FTAIL	line type	char(5)	‘FTAIL’	marks end of file
	line number File Line Identifier	number(10) Numeric ID(10)	Sequential number Created by program.	unique line identifier ID of current line being created for output file.
	file record counter	number(10)		number of FDETL lines in file

Output Specifications

A reject file with the exact same format as the input file will be created, containing any rejected records from the input file. If no records are rejected, this file will consist only of an FHEAD and an FTAIL line.

Scheduling Considerations

This program should be run after the lcmt798 script

Run in Phase II (along with lcupld)

Restart / Recovery

This program will be restartable but not threadable.

Letter of Credit Upload [lcupld]

Design Overview

The purpose of this new Letter of Credit upload program, which will run nightly, is to upload confirmations, drawdowns, and bank fees and charge information returned from an Issuing Bank(s) regarding Letter of Credits. This program should run during the standard upload phase of the batch cycle. The information will be read in using the standard Retek flat file format. A translation script will have previously converted the information to a standard flat file from multiple files formatted in MT 730, which is one of several S.W.I.F.T. formats used by banks to transfer financial information regarding LCs. S.W.I.F.T. formats are used when transmitting financial information and are a globally recognized set of standards. After this program has processed a confirmation/charge/drawdown, the appropriate tables will be updated; a confirmation will update the LC to confirmed status and drawdowns and bank charges will write the appropriate records to the LC_ACTIVITY table.

The lcupld program needs to be modified to insert total amount and currency information into the lc_activity table.

Previously, this program was used to upload confirmations, drawdowns, bank fees, and charge information. The drawdowns and charges will now be handled by lcmt798 (Perl script) and lcup798.pc (batch program); lcupld.pc will only be used for LC confirmations.

Affected Tables

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
LC_HEAD	Yes	Yes	No	Yes	No
LC_ACTIVITY	Yes	No	Yes	No	No

Scheduling Constraints

Processing Cycle: Standard Upload Phase (II)

This program must be run after the lcmt730 Perl script, as it uses the output file from that script. It will be run in Phase 2 along with other upload programs

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

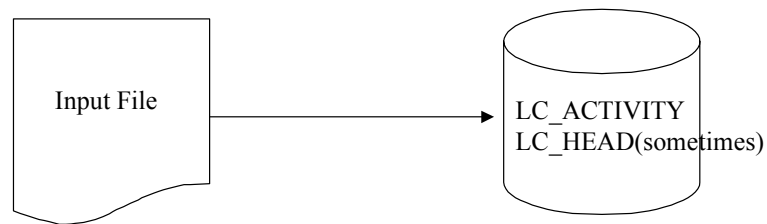
Threading Scheme: No threading due to low volume.

Restart Recovery

Restart/recovery for this program is set up at the individual FDETL record. Although there may be more than one FDETL record for a given LC, they will each be processed as a separate entity.

The recommended commit counter setting is 1000 records (subject to change based on experimentation).

Program Flow



Shared Modules

CURRENCY_SQL.CONVERT

currency_sql.convert will no longer be used.

Function Level Description

Init():

This function performs preliminary processing and populates global variables. It will call `retek_init()` to start up restart-recovery and bring back the bookmark string in case of a restart. It will also populate `vdate`, to be used when update the `lc_head.confirmed_date`. Create and open a reject file for write. Open input file for read. Call `get_record` to read the FHEAD into the `header_struct`.

Process():

This function drives the processing of the program. Call the function `retek_get_record` to read the FDETL line into the `detail_struct`. If the record type returned is FTAIL, exit the loop. If the record type returned is other than FDETL or FTAIL then return a fatal error (wrong record type). If the record type is FDETL then call the function `validate_variables()`, passing in the FDETL variables, to validate and then insert the variables parsed above into the correct fields on the LC_ACTIVITY table.

If the `upload_type = 'L'` (Confirmation of an LC), call `update_lc_head()`. If `validate_variables()` return a non_fatal error, call `retek_write_rej()` to write reject records.

After reading and processing `commit_max_ctr` number of Letter of Credit acknowledgement records, call `retek_force_commit()` to perform intermittent commit for restart/recovery.

Final():

This function will call `retek_close()` to wrap up restart/recovery, populate the final error message, if necessary, and close the input and reject files.

Init_array():

This function initializes the size of the array used for holding the input records at the size of the restart max counter on `restart_control`.

Validate_variables():

This function basically consists of a number of function calls to validate the variables that will later, if they pass validation, be inserted into LC_ACTIVITY.

Call `check_upload_type()`, passing in the `upload_type` variable.

Call `check_lc_refs()`, passing in the `upload_type`, the `bank_lc_id`, the `lc_ref_id`, and the `issuing_bank` variables. Fetch back `lc_head.rowid` if the record is valid.

Call `check_date()`, passing in the `activity_date`.

Call `check_amount()`, to ensure it is all numeric, passing in the amount.

Call `check_currency()`, passing in the `lc_ref_id`, the `currency_code` and the amount.

`Check_upload_type()`:

This function takes in the `upload_type` and ensures that it is 'B', 'D', or 'L'; reject record if it is not. Note: these three codes are being used temporarily until it is confirmed that these are the only three types of MT 730 that will be coming in.

`Check_lc_refs()`:

This function takes in the `upload_type`, `bank_lc_id`, the `lc_ref_id` and the `issuing_bank`. If the `upload_type` is 'B' or 'D', then check if both the `bank_lc_id` and the `lc_ref_id` are valid and exist on the same record on `lc_head` for a 'C' onfirmed Letter of Credit. If the `upload_type` is 'L', then check if only the `lc_ref_id` exists on `lc_head` (as the `bank_lc_id` is being sent with this confirmation and will not be present on the table until after this program runs), the function will also check `lc_head.issuing_bank` with the passing in `issuing_bank`. Reject the record if the check fails, return `lc_head.rowid` if the check passes.

`Check_date()`:

This function check that the date is a valid date (that the date exists, 063199 is an example of a date that would not be valid as it does not exist) by calling `valid_date()`, a library function. Note: Not sure if we insert the date sent in the MT 730 into `lc_activity` or use the `vdate` instead?

`Check_amount()`:

This function takes in the amount and calls the library function `valid_all_numeric()` to ensure that the amount is all numeric. If not reject the record.

`Check_currency()`:

This function takes in the currency and the amount and checks if it exists on the CURRENCY table. If it exists on the currency table, check if it matches the `LC_HEAD.CURRENCY_CODE` for that `lc_ref_id`. If the currency does not match the currency on `lc_head` then use `CURRENCY_SQL.CONVERT` to convert the amount to the `LC_HEAD` currency. Pass in the amount, the currency, the currency off of `lc_head` as the `I_currency_out`, the `I_cost_retail_ind` as 'C', and then NULL for the last four input variables. Reject the record if the package fails.

Note: The `I_exchange_type` will perhaps be entered as 'LC', although this logic is not in place right now.

Update_lc_head():

This function will updates the confirmed_date of the Letter of Credits to vdate, the bank_lc_id to the input value if the upload_type ='L'. This is by rowid.

Insert_records():

This function will insert all the valid records to LC_ACTIVITY through activity_struct array.

In the prototype section, include variable names in the functions.

Throughout the program, remove unused variables (that came from fields on the input file that have been removed).

Since we will only ever be getting confirmations, we will remove the upload type, check_amount, and check_currency from the file and the program. The update to the lc_head table should always be done. The check_upload_type function should be removed. The code in check_lc_refs that does stuff if the upload type is 'B' or 'D' can be removed.

For each confirmation received, we need to insert a record into the lc_activity table that contains the total amount on the letter of credit (from lc_head), the currency code (from lc_head) and the exchange rate (from lc_head). The trans_code should be inserted as 'L' and the activity date should come from the SWIFT file.

Since the amount will come form the lc_head table, it will never need to be converted and the call to CURRENCY_SQL.CONVERT can be removed.

I/O Specification

Input file:

Original Version:

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence Number	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	LCUP	Identifies file as 'Letter of Credit Upload'

Record Name	Field Name	Field Type	Default Value	Description
	File Create Date	Char(14)	create date	current date, formatted to 'YYYYMMDDHH24MISS'
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Sequence Number	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Number(10)	sequence number	Used to force unique file check.
	Sender's Reference	Char(16)	lc_head.bank_lc_id	The LC number that the bank assigns to a Letter of Credit. SWIFT allows for up to 16 characters.
	Receiver's Reference	Number(8)	lc_activity.lc_ref_id	The LC number that Retek assigned to the Letter of Credit.
	Upload Type	Char(1)	lc_activity.transaction_code	Determines what kind of message this individual FDETL is: L = Confirmation of LC B = Bank Charge/Fee D = Drawdown
	Date of Message Being Acknowledged	Char(14)	lc_activity.activity_date	If the upload type is 'L' then this date will match the date MT 700 date of formatted to 'YYYYMMDDHH24MISS'
	Currency of Charge	Char(3)	currency code	Should match lc_head.currency_code for that LC.

Record Name	Field Name	Field Type	Default Value	Description
	Amount of Charge	Number (20,4)	lc_activity.amount	If method of payment = 'D', the amount that will be debited from the credit line (the LC) either for bank charges or drawdowns.
	Amount of Bill	Number (20,4)		If method of payment = 'B' then do we have some sort of field like this. . . that is processed somewhere else besides LC_ACTIVITY.
	Account With Bank	Char(10)	lc_head.issuing_bank	
	Comments	Char(2000)	lc_activity.comments	
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Total number lines	Number(10)	Specified by external system	Total number lines in file

Changed Version:

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence Number	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	LCUP	Identifies file as 'Letter of Credit Upload'

Record Name	Field Name	Field Type	Default Value	Description
	File Create Date	Char (14)	vdate	date file was written by external system 'YYYYMMDDHH24MISS' format
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Sequence Number	Number(10)	specified by external system	Line number of the current file
	Sender's Reference	Char(16) (was 15)	lc_head.bank_ld_id	The LC number that the bank assigns to a Letter of Credit.
	Receiver's Reference	Number(8)	lc_activity.lc_ref_id	The LC number that Retek assigned to the Letter of Credit.
	Date of Message Being Acknowledged	Date (char 8)	lc_activity.activity_date	If the upload type is 'L' then this date will match the date MT 700 date of issue (which we have not resolved between being the vdate or the lc_head.application_date) 'YYYYMMDD' format
	Comments	Char(2000)	lc_activity.comments	Need to truncate? This field will probably be a concatenation of the following SWIFT fields: 71B – Charges, 72 – Sender information.
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Total number lines	Number(10)	Specified by external system	Total number lines in file

Reject File

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will need to be created by the letter of credit upload module and a reject line counter will be required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and trailer record and the file line count will be equal to 0.

The reject filename should also be specified as a runtime parameter.

Technical Issues

N/A

Bill of Lading Upload [lifbolup]

Design Overview

This program will format information originating from the Nautilus Bill of Lading files. The Nautilus files, BOL Upload Header, BOL item upload, and BOL Container Item Upload are SQL Loaded into staging tables: lif_bol_head, lif_bol_item, and lif_bol_container_item. This program will read from the staging tables and create a standard formatted file for Retek's tsfoupld program to process.

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: This program should be run after uploading the bill of lading (BOL) information from Nautilus and SQL Loading the files into the staging tables.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

None.

Program Flow

N/A

Function Level Description

Driving cursor:

```
Select lbh.dc_location,
       lbh.bol_number,
       NVL(SUBSTR(lbh.carrier,1,20),''),
       NVL(lbh.scheduled_arrival_date_time, ''),
       lbc.container_id,
       SUBSTR(lbci.item_id,1,13),
       lbc.distribution_number,
       lbci.unit_quantity*pi_implied_dec,
       lbc.destination_id
FROM lif_bol_head lbh,
     lif_bol_container lbc,
     lif_bol_container_item lbci
WHERE lbh.bol_number = lbc.bol_number
      AND lbc.bol_number = lbci.bol_number
      AND lbc.container_id = lbci.container_id
      AND lbh.dc_location = lbc.dc_location
      AND lbc.dc_location = lbci.dc_location
order by lbh.bol_number, lbci.distribution_number desc;
```

Descending will allow us to fetch allocations before shipments

The program will select the data from the BOL temp tables and write the records to the RMS header and detail flat files.

In determining which fields in the RMS files must be populated, if the distribution number is prefixed by an 'A' we are uploading a cross dock allocation. The allocation number on the detail file must be populated. If the distribution number is not prefixed with an 'A', we are uploading a transfer initiated in RMS, and the distribution number is the shipment number. The 'A' will be stripped from the distribution number when uploading.

Init()

Open output file.

Write FHEAD

Process()

Loop through records found in the driving cursor.

Call get_quantity().

If the BOL number changes from what it was previously:

 Write a header record

If the distribution number is prefixed with an 'A':

 If the bol number changes, then

 Call get_regular_to_loc() to retrieve the allocation location type.

 elseif the distribution number is not prefixed with an 'A'

 Call ship_direct().

 If the shipment is a ship direct, customer order, then

 Call get_ship_direct_to_loc().

 Else

 Call get_regular_to_loc()

 End if;

 End if;

Write a header record when the bol number changes for the new bol number.

Also, write a trailer record for the previous bol number. Call write_Thead() and write_Ttail().

If the record is a shipment, then a header record is written for each new shipment number, regardless of the bol number. For allocations that are processed, only one header record is written.

Call `write_tdetl()` to write the detail information to the flat file.

The output file will have the following format:

FHEAD

THEAD for first BOL

TDETL's for all allocations within first BOL

TTAIL

THEAD for first shipment within first BOL

TDETL's for each detail record within this shipment

TTAIL

<repeat THEAD,TDETL's,TTAIL for each shipment within first BOL>

<repeat for subsequent BOL's>

Within a BOL, all allocations appear as details under one header. In contrast, shipments each have their own header record.

Call `Write_FTAIL()`

`Win_skus_uom()`

Fetch UOM from `win_skus` table.

`Rag_style_uom()`

Fetch UOM from `rag_style` table.

`Packhead_uom()`

Fetch UOM from `packhead` table.

`Get_quantity()`

It is assumed that both `shipped_weight` and `unit_quantity` fields will be populated from RDM.

Thus, UOM of the item corresponding to this BOL should be determined.

An item is `catch_weight` when its UOM is anything other than 'EA'. In order to determine the UOM of an item, the following columns need to be used:

`RAG_STYLE.STANDARD_UOM` for fashion items,
`WIN_SKUS.STANDARD_UOM` for staple items and
`PACKHEAD.STANDARD_UOM` for pack items.

(DESC_LOOK.SYSTEM_IND will display 'f' for fashion item, 'S' for staple item and 'P' for pack item.)

If the UOM of the corresponding item is 'EA' then unit_quantity field should be used for output file. Any item with a UOM other than 'EA' is assumed to be a catch weight item, therefore shipped_weight should be used to write to the transfer quantity field.

Write_FHEAD()

Write the FHEAD record to the file.

Write_THEAD()

Write the THEAD record to the file.

Bol number = DC + BOL

If it is a shipment, then the distribution number equals the shipment number. Otherwise, the shipment value is null.

Write_TDETL()

Write the TDETL record to the file.

If it is a shipment, then the allocation value is NULL. Otherwise, populate the allocation value with the distribution number.

Write_TTAIL()

Write the TTAIL record to the file.

Write_FTAIL()

Write the FTAIL record to the file.

Get_regular_to_loc()

Set location type variable equal to 'ST' or 'WH' whether the location is a store or warehouse.

Ship_direct()

Check if the shipment is a ship direct, customer order.

Get_ship_direct_to_loc()

If a record is a shipment, and if the shipment is scheduled for direct delivery to the customer, replace the destination location and loc_type, with the location and loc_type stored on the shipment table (presumably, the location of the selling store).

If location type = 'S', then

Set the location type = 'ST'

If location type = 'W' then

Set the location type = 'WH'

Write_files()

Write bill of lading information to the standard flat file.

Final()

Write FTAIL, TTAIL

Delete from the temp tables: lif_bol_head, lif_bol_container,
lif_bol_container_item.

Close output file

I/O Specification

Output File

The output file should be accepted as a runtime parameter at the command line.

Note: There is a space between fields in the RLS flat file format.

Output File (in format needed by RMS tsfoupld.pc program)

Record Name	Field Name	Field Type	Default Value	Description
File Header	File type record descriptor	Char(5)	FHEAD	Identifies the file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	TSFO	Identifies file as 'Transfer OUT'
	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 Sequence	Number(10)	specified by external system	Line number of the current file

Record Name	Field Name	Field Type	Default Value	Description
	Transaction Set Control Number	Number(14)	specified by external system	used to force unique transaction check
	Transaction Date	Date	specified by external system	date the transfer was created in external system
	From Location Type	Char(2)	ST - store WH - warehouse	specifies the type of location sending items
	From Location Value	Number(4)	location identifier	Specifies the sending location id number
	To Location Type	Char(2)	ST - store WH - warehouse	specifies the type of location receiving items
	To Location Value	Number(4)	location identifier	Specifies the receiving location id number
	Shipment Number	Number(10)	Retek shipment number	specifies the Retek shipment cross-reference
	External shipment	Char(15)	External shipment number	specifies external shipment number; will be CARTON when transferring cartons
	Courier	Char (20)	Courier used to ship order	
	Scheduled Arrival Date	Date	shipment.ext_arr_date	arrival date
	Number of boxes	Number(4)		Number of boxes in this transfer
	BOL Number	Number (13)	BOL Number	Bill of Lading number
Transaction Detail (Item)	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Number(14)	specified by external system	used to force unique transaction check

Record Name	Field Name	Field Type	Default Value	Description
	Detail Sequence Number	Number(6)	specified by external system	sequential number assigned to detail records within a transaction
	Item Type	Char(3)	UPC SKU	item type will be represented as a UPC or SKU
	Item Value	Number(16)	item identifier	the id number of a SKU or UPC
	Supplement	Number(5)	supplemental identifier	used to further specify the id of an UPC item
	Allocation Number	Char(6)	allocation identifier	Retek allocation number attached to the transfer
	Inventory Status	Number(2)	inventory status of item	used to indicate the type of non-salable merchandise transferred in an 'NS' transfer
	Carton Number	Varchar(20)	Container ID	Container identifier
	Transfer Quantity	Number(12)		number of units to be transferred of the given item (*10000—4 implied decimal places)
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Sequence	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 Sequence	Number(10)	specified by external system	Current line number
	Number of transaction lines	Number(10)	specified by external system	total number of lines in file, excluding FHEAD and FTAIL

Error File

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

N/A

Inventory Adjustment Conversion [lifinvup]

Design Overview

This program converts the Nautilus inventory adjustment upload file into a standard flat file specified by Retek's inventory adjustment upload program. This includes creating a standard flat file (ie. adding FHEAD and FTAIL), adding necessary default values, translating the reason code sent from RLS, and ignoring RTV information that is passed in the inventory adjustment file.

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: This program should be run before the invaupld.pc and rtvupld.pc program and after the Nautilus inventory adjustment upload program, inv_adj_upload.sh.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Program Flow

N/A

Function Level Description

Init()

Functional details:

Open the output file and create file header information. The system date is selected for the file create date and the transaction date used in the output file header and transaction detail respectively.

Call restart recovery init function.

Call function to write output file header information (ie. write_FHEAD())

Process()

Functional Details:

This function should write transaction detail records (TDETL) to the output file for each inventory adjustment sent by Logistics in the input file. The function should fetch the first record from the input file. The field information should be parsed into variables. Other default information, such as location type, item type, UPC supplement, and inventory status should also be formatted and put into variables to be loaded into the TDETL record.

If the reason code = 1 or 80, then

Set the inventory status field = 3. (ie. unavailable inventory adjustment).

 If reason code = 1, then

 Change the quantity signage to a negative number

 elseif reason code = 80, then

 Change the quantity signage to a positive number

 Else

 Set the inventory status field = 0

End if;

End if;

If reason code = 90, then

Do not process.

The reason code will be sent up from Nautilus and populated in RMS' reason code field on the flat file. However, if the reason code = 90, then this record will not be sent to RMS (ie. it will not be written to the TDETL record). Reason code = 90 indicates an RTV upload. RTVs will be handled separately from the inventory adjustment upload program and therefore will not need to be processed in this program.

Write_FHEAD ()

Write the standard header record required by RMS inventory adjustment upload program.

Write_TDETL()

Write the detail record required by RMS inventory adjustment upload program.

Write_FTAIL()

Write standard tail record required by RMS inventory adjustment upload program.

Final()

Call restart close.

Call function to write output file trailer information (ie. write_FTAIL())

I/O Specification

The input file from Nautilus and the output file should be accepted as a runtime parameter at the command line.

Output File (in format needed by RMS inventory adjustment upload program)

Record Name	Field Name	Field Type	Default Value from RLS	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	INVA	Identified file as 'Inventory Adjustments'
	File Create Date	Date	sysdate	date file was written by external system
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Number(14)	specified by external system	used to force unique transaction check
	Transaction Date	Date	sysdate	date of the inventory adjustment

Record Name	Field Name	Field Type	Default Value from RLS	Description
	Location Type	Char(2)	WH - warehouse	specifies the type of location
	Location Value	Number(4)	Location(D C)	specifies the location id number
	Item Type	Char(3)	SKU	item type will be represented as an UPC or an SKU
	Item Value	Number(13)	SKU item id	the id number of a SKU or UPC
	UPC Supplement	Number(5)	N/A	used to further specify the id of an UPC item
	Inventory Status	Number(2)	0 or 3	inventory status code as specified in the inventory status code table - 0 for total stock on hand adjustments or 3 for Nautilus troubled msde.
	Adjustment Reason	Number(2)	reason code	adjustment reason code as specified in the adjustment reason code table
	Adjustment sign	Character(1)	+, -	sign of adjustment qty
	Adjustment Quantity	Number(12, 4)	Unit quantity adjustment	number of units to be adjusted of the given
Transaction Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file

Record Name	Field Name	Field Type	Default Value from RLS	Description
	File Line Count	Number(10)	total transaction detail	sum of all transaction lines, including file header and trailer

Technical Issues

N/A

Receiving Conversion [lifrcvup]

Design Overview

This conversion program will convert the Nautilus receipts upload file into a Retek standard flat file that can be processed by the tsfparse.pc batch program. Tsfparse.pc will then split the one input file into 3 separate receiving files (ie. transfers, PO, cartons).

The Nautilus' receipts upload program will produce one flat file that will be SQL Loaded into a staging receipt table, lif_receipts. This conversion program will sort and process the records found on the staging table and write the required data to a flat file in the format required by the tsfparse.pc program.

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: This program should run before tsfparse.pc and after the SQL loader, receipts_upload.ctl.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

No restart/recovery required.

Main cursor:

```
SELECT location,  
       po_nbr,  
       item_id,  
       DECODE(SIGN(unit_qty),-1, 'N', 'P'),  
       ABS(unit_qty * 10000),  
       tran_type,  
       TO_CHAR(receipt_date, 'YYYYMMDDHH24MISS'),  
       receipt_nbr,
```

```
        asn_nbr,  
        carton_id  
FROM lif_receipts  
ORDER BY location,  
        receipt_nbr;
```

Program Flow

N/A

Function Level Description

Main()

write FHEAD record to output file

Process()

Loop through the records retrieved from receipts staging table.

 A THEAD record will be written for each receipt number.

 If the po number field is NOT prefixed with a 'P', then

 Set the supplier and order number fields equal to NULL, because the receipt is from a transfer shipment received by the warehouse.

 Else the po number field on the input file is prefixed with a 'P'. This is a PO receipt

 Set the order number field equal to the po number on the input file.

 If the ASN number field, which is composed of the Retek the PO number and shipment number (in that order), is populated in the Logistics file, then strip off the shipment number and populate the output file shipment number field. Also, populate the asn number field on the output file.

 Call get_supplier () function.

 End if;

decode adjustment flag

call write_tdetl() function.

Check if receipt number changes. If receipt number is same, then just add detail records (ie. TDETL) to output file. If receipt number, or location, or shipment changes, then write TTAIL to close off the previous receipt and write a new THEAD record to output file.

Get_supplier()

get the order's supplier from ordhead.

Write_fhead ()

Write the FHEAD record to the output file.

Write_thead()

Write the THEAD record to the output file.

Write_tdetl()

If transfer (ie. PO field on input file does NOT have 'P' prefix) receipt, then the inventory status field should be written to the output files as NULL.

If carton number is sent in Logistics file, then populate the carton number field. This will be used for carton Asn receiving.

If the qty > 0, then sign = 'P', else sign = 'N'. Put the absolute value of quantity in the output file (ie. remove the negative or positive sign).

Write the TDETL record to the output file.

Write_ttail()

Write the TTAIL record to the output file.

Final()

Delete receipt from the staging table when all processing is complete.

write FTAIL record to output file

I/O Specification

Output File

The input file, receipts.dat should be accepted as a runtime parameter at the command line.

Output File (in format needed by RMS tsfparse.pc program)

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)	RCPT	Identifies file as 'Receipt' file
	File create date	Date	Create date	Date file was written by external system (YYYYMMDDHHMMSS)
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	Number(14)	Specified by external system	Used to force unique transaction check
	Transaction date	Date	Specified by external system	Date the transfer was created created in external system (YYYYMMDDHHMMSS)
	Location type	Char(2)	WH (hardcode)	Specifies the type of receiving location
	Location value	number(4)	Location (DC)	Specifies the receiving location id number

Record Name	Field Name	Field Type	Default Value	Description
	Adjustment flag	char(1)	Transaction Type (A – adjustment, R – Receipt)	Y – adjustment, N – not adjustment. Transfers will always be ‘N’. Used for receiver error adjustment for common receiving layout.
	Order number	number(8)	PO Number	Retek order number
	Shipment number	number(10)	PO Number for transfer (ie. shipment #) ASN number for ASN (ie. strip off shipment from ASN_nbr)	Retek shipment number When a transfer to the DC is sent (via tsfcdcdnld), then the PO_nbr = shipment. If ASN is sent , then PO+shipment number is sent. We can strip off the shipment number.
	Supplier identifier	number(10)	PO’s supplier	Retek supplier number
	Vendor ASN	char(20)	NULL	Vendor shipment number
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

Record Name	Field Name	Field Type	Default Value	Description
	Transaction set control number	Number(14)	Specified by external system	Used to force unique transaction check
	Detail sequence number	Number(6)	Specified by external system	Sequential number assigned to detail records within a transaction
	Item type	char3)	SKU (hardcoded)	Item type represented as UPC, SKU, VPN
	Item value	number(30)	Item id	Item identifier
	Supplement	number(5)	NULL	Supplemental identifier
	Carton number	number(20)	Carton number	Specifies the unique UCC – 128 Code for a carton
	Inventory status	number(2)	NULL	Used to indicate the type of non-salable merchandise transferred in an 'NS' transfer
	Receipt Quantity Sign Flag	char(1)	P or N	P – positive, N – Negative. Always positive for transfer (file layout common to receiving which can have pos. or neg. adjustments)

Record Name	Field Name	Field Type	Default Value	Description
	Receipt quantity	number(12, 4)	Unit quantity	Number of units received of the item.
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

Technical Issues

N/A

Return to Vendor Upload [lifrtvup]

Design Overview

This program will format information originating from the return to vendor file. The Nautilus file, `rtv_upload.dat`, is SQL Loaded into a staging table: `lif_rtv`. This program will read from the staging tables and create a standard formatted file for Retek's `rtvupld.pc` program to process.

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: This program should be run after uploading the `rtv` information from Nautilus and after SQL Loading the files into the staging tables. It should run before `rtvupld.pc`

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

No restart/recovery, except for error handling.

Main cursor:

```
SELECT location,
       rtv_order_no,
       SUBSTR(item_id, 1, 13),
       rtv_auth_id,
       unit_qty,
       supplier
FROM lif_rtv
ORDER BY location,
       rtv_order_no;
```

Program Flow

N/A

Function Level Description

Init()

The output file should be opened, then file header information should be written.

The vdate is selected from the period table for the file create date used in the output file header.

Call restart init.

Write output file header (FHEAD) information

Process()

Initialize the RTV number variable to NULL.

Loop through the records found on the lif_rtv table. Array processing should be used to fetch the records from lif_rtv table.

If RTV order number changes or if the location changes. (ie. one THEAD for each rtv number)

Write TTAIL for previous RTV

Set default values for THEAD record.

Write THEAD record.

Set default values for TDETL record.

Write TDETL record to the output file for each sku in the RTV.

End If;

End loop;

final()

Close output file.

Delete from lif_rtv table if all no failures occurred.

Write output file trailer (FTAIL) information.

Call restart close.

I/O Specification

Output File

The output file should be accepted as a runtime parameter at the command line.

RTV upload file:

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	RTV	Identifies file as 'Return to Vendor'
	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 Sequence	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 transfer was created in external system
	Supplier Identifier	Char(10)	Vendor_nbr	Supplier reference identifier as identified within Retek

Record Name	Field Name	Field Type	Default Value	Description
	Return Authorization	Char(12)	Rtv_auth_nbr	Supplier return authorization number
	RTV Location Type	Char(2)	WH – warehouse	specifies the type of location returning items
	RTV Location Value	Char(4)	Location id	specifies the returning location id number
	Freight	Char(20)	NULL	Freight cost associated with the RTV in supplier's currency
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence	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
	Detail Sequence Number	Char(6)	specified by external system	sequential number assigned to detail records within a transaction
	Item Type	Char(3)	SKU	item type will be represented as an UPC, an SKU
	Item Value	Char(13)	item id	the id number of a SKU or UPC

Record Name	Field Name	Field Type	Default Value	Description
	Supplement	Char(5)	NULL	used to further specify the id of an UPC item
	Shipment	Char(10)	NULL	Cross reference to retek shipment number for RTVs associated with QC.
	Reason	Char(6)	'W'	Reason for the return: Q - QC failed U - Unavailable inventory W – warehouse initiated RTV from the RTVR code on codes table.
	Return Quantity	Char(12)	Unit qty	number of units returned of the given item
	Unit Cost	Char(20)	NULL	assigned cost value of inventory to be returned
	Inventory Status	Char(2)	NULL	Cross reference to Retek Unavailable inventory status number.
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file

Record Name	Field Name	Field Type	Default Value	Description
	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 Sequence	Number(10)	specified by external system	Line number of the current file
	File Transaction Line Count	Number(10)	sum of all transaction lines	total of all records less file head and tail records

Technical Issues

N/A

Stock Upload Conversion [lifstkup]

Design Overview

This program converts the Nautilus inventory balance upload file into the Retek standard flat file for stkupld.pc to process.

This program verifies that the inventory data is for the requested cycle count and warehouse before proceeding. Other data needed for the RMS flat file will be obtained from RMS tables and inserted in the RMS flat file.

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: This program should run before stkupld.pc and after the Nautilus inv_bal_upload.sh program.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Program Flow

N/A

Function Level Description

Init()

Obtain the cycle_count from tables stake_head and stake_location by matching the warehouse location and the stocktake_date with combination of the Nautilus input file, location(DC) and the transaction date/time (date portion only).

If no match can be made, the program will terminate.

Call restart init

Write FHEAD record.

Process()

Loop through and read each line from the Nautilus file.

Format the data for the output file.

Call write_fdetl() function.

Write_fdetl()

Call check_catch_weight() function

Write a transaction record (FDETL) to the Retek flat file for each line found in the Nautilus flat file.

Check_catch_weight()

Called within write_fdetl to check if item is check weight Copies total weight value if it is a check weight item, otherwise copies total units.

Final ()

Write a file trailer record (FTAIL).

Close files

Call restart close

Write status to log file

I/O Specification

Input File

The input file should be accepted as a runtime parameter at the command line.

Record Name	Field Name	Field Type	Description
File Header	file type record descriptor	Char(5)	hardcode 'FHEAD'
	file line identifier	Number(10)	Id of current line being processed., hardcode '000000001'
	file type	Char(4)	hardcode 'STKU'
	file create date	Date(14) YYYYMMDD HHMISS	date written by convert program
	stocktake_date	Date(14) YYYYMMDD HHMISS	stake_head.stocktake_date
	cycle count	Number(8)	stake_head.cycle_count

Record Name	Field Name	Field Type	Description
	loc_type	Char(2)	hardcode 'WH'
	location	Number(4)	stake_location.wh
Transaction record	file type record descriptor	Char(5)	hardcode 'FDETL'
	file line identifier	Number(10)	Id of current line being processed, internally incremented
	item type	Char(3)	hardcode 'SKU'
	item value	Number(13)	item id
	supplement	Number(5)	NULL
	inventory quantity	Number(12,4)	total units or total weight
	location description	Char(30)	NULL
File trailer	file type record descriptor	Char(5)	hardcode 'FTAIL'
	file line identifier	Number(10)	Id of current line being processed, internally incremented
	file record count	Number(10)	Number of detail records.

Technical Issues

N/A

Location Download [locdnld]

Design Overview

This new program will process records from a staging table, lif_location, and write Location (Ship Destination) information to a file for the Nautilus' Ship Destination Download program to process. This download program will create a Retek Standard flat file, which will contain needed location information. A conversion program will then restructure the format of the flat file and create a ship destination file according to Nautilus' Ship Destination Download flat file specifications.

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: This program does not have any scheduling constraints. It can be run multiple times during a batch schedule.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

The Logical Unit of Work for the program will be based on retrieving a single row from the lif_location table.

```
select l.location,  
       l.loc_type,  
       l.action_type,  
       nvl(s.store_name,nvl(w.wh_name,' ')) location_name,  
       nvl(s.store_add1,nvl(w.wh_add1,' ')) add1,  
       nvl(s.store_add2,nvl(w.wh_add2,' ')) add2,  
       nvl(s.store_city,nvl(w.wh_city,' ')) city,  
       nvl(s.state,nvl(w.state,' ')) state,  
       nvl(s.store_pcode,nvl(w.wh_pcode,' ')) pcode,
```

```

        nvl(s.phone_number,'') phone_number,
        nvl(s.fax_number,'') fax_number,
        nvl(s.store_mgr_name,'') store_mgr_name
    from lif_location l,
        store s,
        wh w
    where l.location = s.store(+)
        and l.location = w.wh(+);

```

Program Flow

N/A

Function Level Description

The program will select the fields from the store and wh tables (based on the information given on the lif_location table) and write the records to the flat file in the Logistics format with header, detail, and tail information.

Init()

Initialize restart recovery.

Open output file.

Write FHEAD

Process()

Open the driving cursor in a FOR loop.

Fetch the data from the lif_location table.

This function should select all location details. Each location (both stores and warehouses) should be written to an output file for the warehouse to upload. Note that with “Delete” records (those records which have an action type of ‘D’), unlike “Add” and “Modify”, it is only necessary to send the location number and the action type information.

Call write_ship_dest() to write the record to the file.

Delete records from lif_location as processed.

Write_ship_dest()

Write location information to the file.

Final()

Write FTAIL

Close restart logic

Close output file

I/O Specification

Input File

The input file should be accepted as a runtime parameter at the command line.

Record Name	Field Name	Field Type	Default Value	Datatype
File Header	File Type Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	Nine leading zeroes: 0000000001	ID of current line being processed by input file.
	File Type Definition	Char(4)	LOCD	Identifies file as 'Location download'
	File Create Date	Date	sysdate	date file was written by external system
Transaction Detail	File Type Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
	Action Type	Char(1)	Created by database trigger.	Action = 'A', Modify = 'M', Delete = 'D'
	Location (DC)	NUMBER(4)	NULL	Code for the DC
	Transaction Date/Time	YYYYMMDDHHMI	sysdate	Date/Time Created on host.
	Destination ID	NUMBER(4)	Store, WH	Ship Destination ID
	Description	VARCHAR(30)	Store_name, Wh_name	Ship Destination Description

Record Name	Field Name	Field Type	Default Value	Datatype
	Shipping Address Line 1	VARCHAR (30)	Store_add1, Wh_add1	Address Line 1
	Shipping Address Line 2	VARCHAR (30)	Store_add2, Wh_add2	Address Line 2
	City	VARCHAR (25)	Store_city, Wh_city	Store or warehouse City
	State	VARCHAR (3)	State	Store or warehouse State
	Zip	VARCHAR (10)	Store_pcode, Wh_pcode	Store or warehouse Zip
	Phone Number	VARCHAR (20)	Phone_number	Store Phone Number, warehouse = NULL
	Fax Destination	VARCHAR (20)	Fax_number	The fax number at the ship destination if store. If warehouse, then NULL.
	Contact Person	VARCHAR (20)	Store_mgr_name	Name of contact at the ship destination, if store. If warehouse, then NULL.
	Distance to Destination	NUMBER(4)	"0" (Zero)	Distance from the DC to the Ship Destination.
	Lead Time	NUMBER(3)	"0" (Zero)	The time in hrs expected for the trailer to arrive at the ship destination from the DC.
	Drop Trailers Accepted Flag	VARCHAR (1)	"N"	Indicates if Drop Trailer is expected at Destination
	Receive Dock Available Flag	VARCHAR (1)	"N"	Indicates if a Receiving dock is available at the destination

Record Name	Field Name	Field Type	Default Value	Datatype
	BOL Upload Type	VARCHAR (4)	"CONT"	"ITEM" = item, "CONT" = container, "BOTH" = item and container.
	BOL Print Type	VARCHAR (4)	"CONT"	"ITEM" = item, "CONT" = container, "BOTH" = item and container, "NONE" = do not print a BOL
	Default Carrier Code	VARCHAR (4)	Code_detail.code table where code = DEFT	Code of the default carrier for destination.
	Default Service Code	VARCHAR (6)	Code_detail.code table where code = DEFT	Code of the default service class for destination.
	Default Route	VARCHAR (10)	Code_detail.code table where code = DEFT	Default route for destination.
	Expedite Carrier Code	VARCHAR (4)	Same code as default carrier	Code of the expediate carrier for destination.
	Expedite Service Code	VARCHAR (6)	Same code as default service	Code of the expedite service class for destination.
	Expedite Route	VARCHAR (10)	Same code as default route	Expedite route for destination.
File Trailer	File Type Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
	File Record Counter	Number(10)	Determined internally	Number of records/transactions processed in current file (only records between head & tail)

Error File

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

N/A

[onordext.pc]

Design Overview

This program calculates the value in cost and retail of items that are on order for the department/class/subclass/location level. This program is the first step in the stock ledger download process to RPP. It calculates the on order cost and retail for all approved orders that have not before dates less than or equal to the planning horizon date. Once the program has calculated the costs and retails, they are inserted into the ON_ORDER_TEMP table. This table is used in the second step of the stock ledger download process -- the Stock Ledger Extract program (stlgdnld.pc).

Important Note: The MAX_BUFFER_NUM #define should be set to a number that will allow safe buffering of component pack items, allocations, and orders at the component sku location level. If it is possible for any of these variable to exceed the value assigned to MAX_BUFFER_NUM, its value must be increased.

Stored Procedures / Shared Modules (Maintainability)

Convert_to_primary – Pro*c library (utils.h / utils.pc).

Input Specifications

Command Line Parameters:

Onordext will calculate the on order cost and retail for a given planning horizon. The planning horizon date will be a command line parameter contained in an input file. The file will also contain information used by stlgdnld.pc.

The file contains 1 line

Field Position: 1 – weekly or historic indicator

2-9 – planning horizon start date

10-17 – planning horizon end date

Onordext userid/passwd input-file

Driving Cursor:

There are two driving cursors in this program. The first deals with staple and fashion items. The second deals with pack items.

/*

The first driving cursor handles on-order amounts for staple and fashion items. The first part deals with non-allocation orders or allocation orders that are not pre-marked

*/

EXEC SQL DECLARE c_driver CURSOR FOR

```

SELECT oh.order_no,
       -999 alloc_no,
       -999 sku, /* needed so the comparison in handle_alloc will not fail */
       sum(ol.qty_ordered - nvl(ol.qty_received,0)),
       os.unit_cost,
       ol.unit_retail,
       ol.location,
       ol.loc_type,
       -999 alloc_loc,
       'N' alloc_loc_type,
       0 alloc_qty,
       dl.dept,
       dl.class,
       dl.subclass,
       oh.exchange_rate,
       oh.currency_code
FROM ordhead oh,
     ordsku os,
     ordloc ol,
     desc_look dl
WHERE oh.status = 'A'

AND ol.qty_ordered > nvl(ol.qty_received,0)

```

```
AND oh.not_before_date <= TO_DATE(:ps_on_order_date,
'YYYYMMDD')

AND oh.order_no = os.order_no

AND os.order_no = ol.order_no

AND os.sku    = ol.sku

AND ol.sku    = dl.sku

AND dl.system_ind != 'P'

AND (oh.pre_mark_ind = 'N' OR
     (oh.pre_mark_ind = 'Y'
      and not exists (select alloc_no
                      from alloc_header ah
                      where ah.order_no = oh.order_no
                           and ol.sku    = ah.sku
                           and ol.location = ah.wh)))

AND oh.order_no > NVL(:ps_restart_order, -999)

AND MOD(oh.order_no, :pi_restart_num_threads) + 1 =
:pi_restart_thread_val

GROUP BY oh.order_no,
         os.unit_cost,
         ol.unit_retail,
         ol.location,
         ol.loc_type,
         dl.dept,
         dl.class,
         dl.subclass,
         oh.exchange_rate,
         oh.currency_code
```

UNION ALL

```
SELECT oh.order_no,
       NVL(ah.alloc_no,-999),
       -999 sku, /* needed so the comparison in handle_alloc will not fail */
       sum(ol.qty_ordered - nvl(ol.qty_received,0)),
       os.unit_cost,
       ol.unit_retail,
       ol.location,
       ol.loc_type,
       DECODE(ad.wh,-1,ad.store,NULL,-999, ad.wh),
       DECODE(ad.wh,-1,'S',NULL,'N','W'),
       sum(nvl(ad.qty_allocated, 0) - nvl(ad.qty_transferred,0)),
       dl.dept,
       dl.class,
       dl.subclass,
       oh.exchange_rate,
       oh.currency_code
FROM ordhead oh,
     ordsku os,
     ordloc ol,
     desc_look dl,
     alloc_header ah,
     alloc_detail ad
WHERE oh.status = 'A'
     AND ol.qty_ordered > nvl(ol.qty_received,0)
     AND oh.pre_mark_ind = 'Y'
```

AND oh.not_before_date <= TO_DATE(:ps_on_order_date,
'YYYYMMDD')

AND oh.order_no = os.order_no

AND os.order_no = ol.order_no

AND os.sku = ol.sku

AND ol.sku = dl.sku

AND dl.system_ind != 'P'

AND ol.order_no = ah.order_no

AND ol.location = ah.wh

AND ol.sku = ah.sku

AND ah.alloc_no = ad.alloc_no

AND oh.order_no > NVL(:ps_restart_order, -999)

AND MOD(oh.order_no, :pi_restart_num_threads) + 1 =
:pi_restart_thread_val

GROUP BY oh.order_no,

ah.alloc_no,

os.unit_cost,

ol.unit_retail,

ol.location,

ol.loc_type,

DECODE(ad.wh,-1,ad.store,NULL, -999, ad.wh),

DECODE(ad.wh,-1,'S',NULL,'N','W'),

dl.dept,

dl.class,

dl.subclass,

oh.exchange_rate,

oh.currency_code

```
ORDER BY 1,2,6,8;
```

```
/*
```

The second driving cursor handles on-order amounts for packitems.

- The first part deals with packs being ordered directly to a store.
- The second part deals with packs being directly ordered to a wh or allocated from a wh with the pre-mark indicator set to 'N'.
- The third part deals with packs being allocated from a wh with the pre-mark indicator set to 'Y'.

```
*/
```

```
EXEC SQL DECLARE c_pack_driver CURSOR FOR
```

```
SELECT oh.order_no,
       -999,
       ol.sku pack_no,
       ol.qty_ordered - nvl(ol.qty_received,0) pack_qty,
       os.unit_cost pack_cost,
       ol.unit_retail pack_retail,
       vpq.sku,
       vpq.qty sku_qty,
       isc.unit_cost comp_cost,
       izp.unit_retail comp_retail,
       ol.location,
       ol.loc_type,
       -999 alloc_loc,
       'N' alloc_loc_type,
       0 alloc_qty,
       dl.dept,
```

```
        dl.class,
        dl.subclass,
        isc.supplier,
        izp.zone_id,
        izp.zone_group_id,
        oh.exchange_rate,
        oh.currency_code
FROM ordhead oh,
     ordsku os,
     ordloc ol,
     desc_look dl,
     v_packsku_qty vpq,
     item_supp_country isc,
     price_zone_group_store pzgs,
     item_zone_price izp
WHERE oh.status      = 'A'

      AND ol.qty_ordered > nvl(ol.qty_received,0)

      AND oh.not_before_date <= TO_DATE(:ps_on_order_date,
'YYYYMMDD')

      AND oh.pre_mark_ind  = 'N'

      AND oh.order_no      = os.order_no

      AND oh.order_no      = ol.order_no

      AND ol.loc_type      = 'S'

      AND os.sku           = ol.sku

      AND ol.sku           = vpq.pack_no

      AND dl.sku           = vpq.sku

      AND isc.item         = dl.sku
```



```

AND oh.supplier      = isc.supplier

AND os.origin_country_id = isc.origin_country_id

AND pzgs.store       = ol.location

AND pzgs.zone_id     = izp.zone_id

AND pzgs.zone_group_id = izp.zone_group_id

AND izp.item         = vpq.sku

AND oh.order_no > NVL(:ps_restart_order, -999)

AND MOD(oh.order_no, :pi_restart_num_threads) + 1 =
:pi_restart_thread_val

UNION ALL

SELECT oh.order_no,

      -999,

      ol.sku pack_no,

      ol.qty_ordered - nvl(ol.qty_received,0) pack_qty,

      os.unit_cost pack_cost,

      ol.unit_retail pack_retail,

      vpq.sku,

      vpq.qty sku_qty,

      isc.unit_cost comp_cost,

      izp.unit_retail comp_retail,

      ol.location,

      ol.loc_type,

      -999 alloc_loc,

      'N' alloc_loc_type,

      0 alloc_qty,

      dl.dept,

      dl.class,

```

```
        dl.subclass,
        isc.supplier,
        izp.zone_id,
        izp.zone_group_id,
        oh.exchange_rate,
        oh.currency_code
FROM ordhead oh,
        ordsku os,
        ordloc ol,
        desc_look dl,
        v_packsku_qty vpq,
        item_supp_country isc,
        item_zone_price izp
WHERE oh.status      = 'A'

      AND ol.qty_ordered > nvl(ol.qty_received,0)

      AND oh.not_before_date <= TO_DATE(:ps_on_order_date,
'YYYYMMDD')

      AND oh.order_no      = os.order_no

      AND oh.order_no      = ol.order_no

      AND os.sku           = ol.sku

      AND ol.loc_type      = 'W'

      AND ol.sku           = vpq.pack_no

      AND dl.sku           = vpq.sku

      AND isc.item         = dl.sku

      AND oh.supplier      = isc.supplier

      AND os.origin_country_id = isc.origin_country_id

      AND izp.item         = vpq.sku
```

```

AND izp.base_retail_ind = 'Y'

AND (oh.pre_mark_ind = 'N' OR
     (oh.pre_mark_ind = 'Y'
      and not exists (select alloc_no
                      from alloc_header ah
                      where ah.order_no = oh.order_no
                           and ol.sku    = ah.sku
                           and ol.location = ah.wh))))

AND oh.order_no > NVL(:ps_restart_order, -999)

AND MOD(oh.order_no, :pi_restart_num_threads) + 1 =
:pi_restart_thread_val

UNION ALL

SELECT oh.order_no,
       NVL(ah.alloc_no,-999),
       ol.sku pack_no,
       ol.qty_ordered - nvl(ol.qty_received,0) pack_qty,
       os.unit_cost pack_cost,
       ol.unit_retail pack_retail,
       vpq.sku,
       vpq.qty sku_qty,
       isc.unit_cost comp_cost,
       izp.unit_retail comp_retail,
       ol.location,
       ol.loc_type,
       DECODE(ad.wh,-1,ad.store,NULL,-999, ad.wh),
       DECODE(ad.wh,-1,'S',NULL,'N','W'),
       nvl(ad.qty_allocated, 0) - nvl(ad.qty_transferred,0),

```

```
        dl.dept,
        dl.class,
        dl.subclass,
        isc.supplier,
        izp.zone_id,
        izp.zone_group_id,
        oh.exchange_rate,
        oh.currency_code
FROM ordhead oh,
        ordsku os,
        ordloc ol,
        desc_look dl,
        v_packsku_qty vpq,
        item_supp_country isc,
        item_zone_price izp,
        alloc_header ah,
        alloc_detail ad
WHERE oh.status      = 'A'

      AND ol.qty_ordered > nvl(ol.qty_received,0)

      AND oh.pre_mark_ind = 'Y'

      AND oh.not_before_date <= TO_DATE(:ps_on_order_date,
'YYYYMMDD')

      AND oh.order_no      = os.order_no

      AND oh.order_no      = ol.order_no

      AND os.sku           = ol.sku

      AND ol.loc_type      = 'W'

      AND ol.sku           = vpq.pack_no
```

```

AND dl.sku          = vpq.sku
AND isc.item        = dl.sku
AND oh.supplier     = isc.supplier
AND os.origin_country_id = isc.origin_country_id
AND izp.item        = vpq.sku
AND izp.base_retail_ind = 'Y'
AND ol.order_no     = ah.order_no
AND ol.location     = ah.wh
AND ol.sku          = ah.sku
AND ah.alloc_no     = ad.alloc_no
AND oh.order_no > NVL(:ps_restart_order, -999)
AND MOD(oh.order_no, :pi_restart_num_threads) + 1 =
:pi_restart_thread_val
ORDER BY 1,2,3,11,13;

```

Output Specifications

ON_ORDER_TEMP will be populated by this program.

ON_ORDER_TEMP

DEPT	NUMBER(4)
CLASS	NUMBER(4)
SUBCLASS	NUMBER(4)
STORE	NUMBER(4)
WH	NUMBER(4)
ON_ORDER_RETAIL	NUMBER(20,4)
ON_ORDER_COST	NUMBER(20,4)

Function Level Description

main():

The standard Retek main() function. Calls init(), process(), and final().

init():

Initialize restart recovery by calling retek_init(). Opens the input file and reads the planning horizon end date. Gets the primary currency from the system_options table.

process():

There are two process loops, one for staple and fashion items and one for pack items.

For each record brought back by the staple and fashion driving cursor:

Convert the cost from the order currency into the primary currency by calling the convert_to_primary function.

Convert the retail from the location currency into the primary currency by calling the convert_to_primary function.

Call the handle_alloc function.

After all the record from the staple and fashion cursor have been processed. The process flag should be flipped to 'P'ack. The handle_alloc function is called to clean up the last allocation.

For each record brought back by the pack driving cursor:

Convert the pack cost from the order currency into the primary currency by calling the convert_to_primary function.

Convert the pack retail from the location currency into the primary currency by calling the convert_to_primary function.

Convert the pack component sku cost from the supplier currency into the primary currency by calling the convert_to_primary function.

Convert the pack component sku retail from the zone/zone_group currency into the primary currency by calling the convert_to_primary function.

Call the handle_pack function.

Handle_insert()

This function adds records to arrays that will be inserted into ON_ORDER_TEMP. First it populated a record into the regular array (non-allocation). Then if the current record is associated with an allocation, a record is added to the allocation array.

Add_insert_record()

This function copies records from a driving cursor record into an insert array. It has two modes, the first if for non-allocation records. The cost and retail are set to the converted cost and retail from the driving cursor times the sku qty from ordloc. The second mode handles allocation records, it sets the cost and retail to the converted cost and retail from the driving cursor times the alloc qty from alloc_detail.

Handle_alloc()

This function sums up the quantities allocated to on alloc_header for each allocation. Once every record associated with a allocation have been summed, it is possible to determine how much of the qty ordered to the allocation warehouse should be assigned to it. If all of the items ordered to it have been allocated, the warehouse is not assigned any of the cost or retail. If items are not fully allocated, the warehouse is assigned the items that were not allocated. Once the qty to give to the warehouse it determined, the handle_insert is function is called for each allocation record.

Handle_pack()

This function prorates the amount and cost ordered through packs to their component items. Every component item ordered through a pack has their total cost and retail summed up. Once all the component items have been summed, the cost and retail of the pack is prorated. The formula cost is: (the pack's cost / the summed component cost) * the individual component sku's cost. The same formula is used for retail.

After the records are prorated they are sent to buffer_alloc().

Buffer_alloc()

This function is used to reorder the records sent to it by handle_pack() before they are sent to handle_alloc(). The handle_pack() function requires that the records be ordered by order_no, pack_no, location, alloc location. Handle_alloc() requires the records to be ordered by order_no, pack_no, location, component sku. This function build groups of order_no/pack_no/location and reorders them by component sku. sort_for_allocation(), get_max_indx(), swap_driv_array(), and copy_info() are utility functions that help sort the driving cursor records.

Once the buffered records have been ordered, they are sent to the handle_alloc() function.

Copy_fetch()

Utility function used by handle_alloc and handle_pack. It totals up the allocated qty for the handle_alloc function. It totals up the components cost and retail (times the pack sku qty) for handle_pack.

Insert_on_order_temp()

This function performs an array insert into the on_order_temp table.

Size_driv_cur_struct()

Allocates memory for the structure used to fetch the driving cursors and used when summing up allocations and pack component items.

Size_insert_struct()

Allocates memory used to perform bulk inserts into the on_order_temp table.

Scheduling Considerations

This program can be run weekly as well as ad hoc (for historic data).

Prepost onordext pre must run before this program.

Restart/Recovery

Logical unit of work (LUW) is a unique order number. This is a non-unique LUW.

It is also split into two sections sku and pack. First all skus on orders are processed. When they are done a pack 'flag' is turned on and the restart order is reset. Then all the packs on order are processed. So all orders are considered twice, once for skus and once for packs.

[orddscent.pc]

Design Overview

This new batch program will find discounts that apply to purchase orders and insert the discounts for the items on the order into the `ordsku_discount` table. For off-invoice discounts, it will actually calculate what the new unit cost should be (apply the discount) and update `ordsku` with this value. An order can be marked to have its costs recalculated even if no deals affect it (if there was a supplier cost change), in which case the `ordsku` table will be updated using the new value of the `unit_cost` from the `item_supp_country` table.

Scheduling Constraints

Processing Cycle: Ad hoc—may be run multiple times a day

Scheduling Diagram: Must be run after `ditinsrt.pc`, `reclsdly.pc`, and `scceext.pc` (all programs that insert records into `deal_calc_queue`). Needs to be run before `edidlord.pc` so that order costs are updated before being sent out.

Threading Scheme: thread by supplier

Restart Recovery

Logical unit of work: order

```
SELECT DISTINCT dcq.order_no,
               dcq.recalc_all_ind,
               dcq.override_manual_ind,
               dcq.order_appr_ind,
               o.supplier,
               o.import_order_ind,
               NVL(o.import_country_id, '-99'),
               TO_CHAR(o.not_before_date, 'YYYYMMDD'),
               o.currency_code,
               o.status,
               NVL(o.exchange_rate, 0),
               s.currency_code
FROM   deal_calc_queue dcq, ordhead o, sups s, v_restart_supplier vrs
WHERE  dcq.order_no = o.order_no
       AND o.status <> 'C' /* don't get closed orders */
       AND s.supplier = o.supplier
       AND vrs.driver_value = o.supplier
       AND vrs.num_threads = :pi_restart_num_threads
       AND vrs.thread_val = :pi_restart_thread_val
ORDER BY dcq.order_no
```

This program has inherent restartability, since records are deleted from deal_calc_queue as they are processed.

Recommended maximum commit counter: low.

Shared Modules

CURRENCY_SQL.CONVERT –convert an amount in deal currency to order currency (or vice versa)

ORDER_SETUP_SQL.LOCK_ORDHEAD –lock an order so program can process it without interference from online users (if the function fails, the order is already being used online and will be skipped by the program)

UOM_SQL.CONVERT –convert a quantity in the order's unit of measure to a quantity in the deal's unit of measure

ELC_CALC_SQL.CALC_COMP –calculate expenses and assessments

ELC_CALC_SQL.CALC_TOTALS - Calculates the Total Estimated Landed Cost values for a particular Item

OTB_SQL.ORD_RECEIVE - updates the OTB table

Function Level Description

init:

Retrieve vdate from the period table (this will be used as the last_calc_date for inserts into the ordsku_discount table).

Retrieve the annual first vs promotional first and earliest to latest/latest to earliest indicators (deal_type_priority and deal_age_priority) from system_options. These determine what order deals should be applied in. Also retrieve the elc_ind from system_options; this will be used when calculating changes to estimated landed cost caused by changes in order unit costs created by discounts.

Restart/recovery initialization

process:

Call size_arrays to allocate memory for the arrays.

Fetch all orders that are waiting to be recalculated (from the deal_calc_queue table—driving cursor; use an array fetch)

For each order, call check_lock to determine if the order is locked by an online user. If it is, skip it (don't process at all). Otherwise, copy it to a new array for further processing.

Call reset_orders to remove any existing ordsku_discount records and reset the unit costs on ordsku to their original values (not including any discounts).

For each order:

Find all deals that are active for the order by calling `get_deals`. This is the set of deals that could possibly affect items in the order; further processing will detect exactly which deals affect which items.

Call `make_ordsku_build` to create records on the `ordsku_discount_build` table for each sku/location on the order, including all hierarchy information. (Organization hierarchy information only needs to be found if deals with some organization hierarchy information present were found, which is why this function isn't called until after the deals have been found). Also fetch the `cost_zone` and `otb_calc_type` for each sku.

If the order is in approved status, call `get_otb_cost` to get the original order costs including ELC (these will be used for the final OTB update).

If `recalc_all_ind` was set to 'Y', supplier cost changes have occurred; call `update_supplier_costs` to get the new unit costs and update the `ordsku` table.

For each of the deals found for the order, find all order items that are affected, find the right threshold values, and insert into the `ordsku_discount` table. This will be done by calling `find_items_for_deal` (which in turn calls subfunctions to get threshold values and do the inserts).

Having built `ordsku_discount` records, take all the off invoice discounts for each item on the `ordsku_discount` table for the order and apply them (call `calculate_cost`) and update `unit_cost` on `ordsku` table (call `update_ordsku`) with the new cost including off invoice discounts.

If the order is in approved status, call `get_otb_cost` to get the final order costs including ELC (these will be used for the final OTB update).

If the order is approved, call `update_OTB` to update OTB costs to account for the cost differences caused by applying the deals.

Delete from `ordsku_discount_build` where the order number is the current order number (call `delete_ordsku_disc_bld`).

(Outside of order loop, for each set of orders fetched)

Call `delete_dcq` to delete processed records from the `deal_calc_queue` table.

Call restart commit logic—do a forced commit after each set of orders. This will also release the lock on those orders.

`check_lock`:

Given an order number, call `ORDER_SETUP_SQL.LOCK_ORDHEAD`. If this function returns false, the order is in use online and should not be processed by `orddsent`. (This has the side benefit that if it returns true, online users will see the order as locked and won't be able to mess with it while `orddsent` is updating it. The lock will automatically be cleared when a commit is done).

`reset_orders`:

Given an array of order numbers, first, update `unit_cost` of all `ordsku` records for these orders (for orders in Approved status, using `ordsku.unit_cost`; others, using `item_supp_country.unit_cost`). Second, find maximum `last_calc_date` from `ordsku_discount` table for orders in approved status. At last, delete all records on `ordsku_discount` for these orders.

`delete_dcq`:

Do an array delete of processed records on the `deal_calc_queue` table.

`get_deals`:

Find all deals that are from the order's supplier or from a supplier hierarchy that includes the supplier, if the deal is at a higher level than supplier, by joining to `item_supp_country` (get these now, even if get too many or some that aren't right country—we'll limit to exact matches later anyway), are approved, and are active for the order (deal start date is on or before the order's not before date and deal close date is on or after the order's not before date). Do not fetch vendor-funded deals (`deal_type = 'M'`) If the deal is order-specific (an order number is given on `deal_head`), only fetch it if the order number matches (that is, order number on `deal_head` should either match the order's order number or be NULL). If the `ord_appr_ind` from `deal_calc_queue` is 'N' (this order was not just approved) and the order status is 'A'pproved, fetch deals whose `deal_head.recalc_approved_orders` is 'Y' or `deal_head.approval_date` is less than or equal to order's `ordsku_discount.last_calc_date`. The deals should be fetched in the order in which they should be applied (create date latest to earliest or earliest to latest, order/promotional/annual or order/annual/promotional).

The cursor should also order by application order within the deal. This will entail

picking the right one of 4 separate cursors:

- earliest to latest, annual before promotional
- earliest to latest, promotional before annual
- latest to earliest, annual before promotional
- latest to earliest, promotional before annual

The following information needs to be fetched:

from deal_head: deal_head.type, currency_code

from deal_detail: deal_id, deal_detail_id, application_order, billing_type,
deal_appl_timing, deal_class, threshold_limit_type, threshold_limit_uom,
threshold_value_type, qty_thresh_buy_item, qty_thresh_buy_qty,
qty_thresh_recur_ind, qty_thresh_buy_target, qty_thresh_free_item,
qty_thresh_free_qty, qty_thresh_free_item-unit_cost, rebate_ind,
rebate_purch_sales_ind

Once the deals have been fetched (use the open_cursor and fetch_cursor functions to use the right cursor), check to see if an order-specific deal was found (if the data was fetched sorted by order number, the first deal in the array would have an order number if any of them do). If it was, it will be the ONLY deal that should be processed for this order; the others should be ignored (but remember that one deal can have several deal_detail components, so there may be more than one record that needs to be processed).

For each deal, call org_level_given to see if the deal contains any organizational hierarchy information (if it doesn't, no need to get organizational hierarchy information for the items on the order later—set a flag).

open_cursor:

Given an ordering scheme, open the appropriate deal info cursor

fetch_cursor:

Given an ordering scheme, fetch the appropriate deal info cursor

org_level_given:

Given a deal_id and deal_detail_id, check to see if there exist any records on the deal_itemloc table that have an org_level that is not NULL.

make_ordsku_build:

Given an order, we need supplier/item/location/country id/all hierarchy info (merch and org) for all items on the order that are due to be processed. This means all items that have NOT been received (ordloc.qty_received = 0 or is NULL). If an item is a pack item: a) vendor packs are treated as single items; no special treatment necessary (leave the pack_no field as null when inserting to ordsku_discount_build) b) buyer packs need to be broken down into their component skus—the component sku goes in the sku field and the pack_no goes into the pack_no field of ordsku_discount_build; the quantity ordered of the pack times the skus pack quantity goes into qty_ordered). Get the item/origin country id off ordsku, locations and order quantities from ordloc, supplier hierarchy information from item_supp_country, then call get_merch_hierarchy to get the merchandise hierarchy and cost zone group information. Do not fetch skus if ordsku.cost_source = 'MANL' and deal_calc_queue.override_manual_ind is 'N'—deal calculations wouldn't affect these skus anyway. Also fetch the unit_cost from ordsku. If you are getting the component skus for a buyer pack, use the order's supplier and the pack's origin country (from ordsku) .

If any deals having deal_itemloc records with non-NULL organizational hierarchy levels were found by get_deals, (call org_level_given to check this) call get_org_hierarchy to get organization hierarchy information for each item/location. Only do this for locations that are stores—warehouses are not part of the organizational hierarchy. (Off-invoice deals are not location specific, so location information wouldn't be needed for them).

Array insert the information into the ordsku_discount_build table.

get_merch_hierarchy:

Given a SKU, get all of its merchandise hierarchy information—dept, class, subclass from the desc_look table, style, color, size1 and size2 from the rag_skus table if it is a fashion item, division and group number from the group and deps tables. Also get the cost_zone_group from win_skus, rag_style, or packhead (depending on the item type), and the otb_calc_type from deps.

get_org_hierarchy:

Given a SKU and a store, find the corresponding organizational hierarchy information—district, region, area, chain from the store, district, region, and area tables.

get_otb_cost:

This function will get the total landed cost for each item on the order in primary currency, summing over all cost zones. If ELC is turned on, this will be done by calling ELC_CALC_SQL.CALC_TOTALS for each zone for each sku. The zones come from cost_zone_group_loc (where the zone_group_id is the item's cost_zone_group_id that we got previously, and cost_zone_group_loc.location matches the location from ordloc. The cost with ELC for each item for each zone is multiplied by the number of items ordered (times the pack qty, if the item is part of a pack) and the sum over all zones is saved as the original OTB cost.

If ELC is turned off, just get the unit cost of the item times the quantity ordered (times the pack qty, if it's a pack item), in primary currency. Only do this stuff (getting cost with ELC) if the `otb_calc_type` for the item is 'C'.

`get_threshold_value`:

Given a `deal_id`, a `deal_detail_id`, a quantity ordered, a flag indicating if the threshold type is quantity or amount, and a rebate indicator:

If a UOM is specified on the `deal_detail` (`threshold_limit_uom` is not NULL) call `convert_uom` to convert the ordered quantity to a quantity in the deal's UOM (the thresholds will be based on this unit of measure, not on the default UOM for the item).

1. if this is NOT a rebate: If the threshold is by amount, convert the quantity to a monetary amount (based on the unit cost from `ordsku`). Return the value from the `deal_threshold` table where the `deal_id` and `deal_detail_id` match and the quantity or amount, depending on the threshold type, falls between the `lower_limit` and the `upper_limit`. The unit cost must be converted to deal currency if necessary to do the lookup and then the final amount must be converted from deal currency back to order currency. (Hopefully the currencies will usually be the same!)
2. if it IS a rebate (`deal_detail.rebate_ind = 'Y'`) get the value from `deal_threshold` for the target level (where the `deal_id` and `deal_detail_id`'s match and the `target_level_ind` is 'Y') (again, it may be necessary to do currency conversions)

`convert_uom`:

Call `UOM_SQL.CONVERT` to convert a quantity in the order's unit of measure into a quantity in the deal's unit of measure. Pass in item, supplier, origin country, quantity ordered, deal UOM; get back quantity in deal UOM (The "from" UOM should be passed into the package as NULL, which will force it to go out and get the default UOM for the item, which is what it's ordered in).

`find_items_for_deal`:

Given a deal (`deal_id` and `deal_detail_id`), see if it applies to any of the items in the current order (compare `deal_itemloc` records to the item/hierarchy information previously obtained for the items on the order, using the `ordsku_discount_build` table). Remember to account for excluded parts of the hierarchy as well as included parts.

If there are multiple records (with the same `deal_id` and `deal_detail_id`) with the same organization level, sum the qty ordered among them and combine into 1 record (for example, you could have an order that had a sku going to multiple locations that were listed separately on the deal; just getting rid of the duplicates would ignore some of the qty ordered)

An application order parameter needs to be associated with each sku/order across all deals (this is NOT the same as `deal_detail.application_order`). This should start at 1 for the first record to be inserted into `ordsku_discount` for a given order/sku and be incremented by one for each new record with the same order and sku (we have been painstakingly fetching things in the right order). This will be used later when the deals are actually applied.

Only apply rebates that apply to purchases (if `deal_detail.rebate_ind = 'Y'` only apply records where `deal_detail.rebate_purch_sales_type = 'P'`, otherwise ignore them)

If a deal applies to multiple items in the order, the threshold value for each item should be chosen based on the total number ordered of ALL the items affected. (If the order is for 5 widgets and 4 thingmabobs and both widgets and thingmabobs are covered by the deal, we want to use 9 as the quantity to look up the threshold value for for both widgets and thingmabobs). Thus, after all skus are fetched and any multiple hierarchy levels removed, the total quantity of all skus affected by the same deal and `deal_detail` must be calculated. This summed quantity is what should be passed into the `get_threshold_value` function.

Given the total quantity ordered for all skus on a deal, call `get_threshold_value` for each sku to find what actual value the discount will have. (Don't call the function for the buy/get free discount type—its threshold info was already fetched from `deal_detail`. For this discount type, the `ordsku_discount.discount_value` must actually be calculated out according to the rules listed in `calculate_costs` and the `discount_type` should be 'Q').

Finally, call `insert_to_discount` to insert a record for each sku into the `ordsku_discount` table. The `location_deal_ind` on `ordsku_discount` should be 'Y' if `deal_itemloc.org_level` is given (a location exists), 'N' otherwise. The `paid_ind` on `ordsku_discount` should be 'Y' if this is an off-invoice deal and `deal_application_timing` is set for PO approval, (`deal_detail.billing_type = 'OI'` and `deal_detail.deal_appl_timing = 'O'`) and 'N' otherwise. The `discount_type` should be the same as the `threshold_value_type` on `deal_detail`. If the item is part of a pack, populate the `pack_no` field on `ordsku_discount`; otherwise leave it as NULL.

Exclusive discounts should be the ONLY discount applied to the sku that they're found for.

If you encounter an exclusive discount:

- Delete any existing records for this sku from `ordsku_discount`
- Update `ordsku.unit_cost` to `unit_cost_init` for this order/sku
- Find the deal value, insert into `ordsku_discount` (as usual)
- Delete all records from `ordsku_discount_build` for this order/sku (so that future fetches for different deals with the same sku won't be applied).

convert_currency:

Given order and deal currencies and an amount in the deal currency, call CURRENCY_SQL.CONVERT to find the corresponding amount in order currency. (or vice versa)

Convert_qty_to_amount:

Given a number of items ordered and the unit cost from ordsku, convert to a currency amount (so can look up on deal_threshold table) (If the deal currency and order currency are not the same, the order unit price must first be converted to be in the deal currency).

insert_to_discount:

Array insert records into the ordsku_discount table

update_ordsku:

Given a SKU, an order number and an cost that includes off-discount calculations, update the unit cost on the corresponding ordsku record.

update_supplier_cost:

For each item on an given order (all records on ordsku), update ordsku.unit_cost and ordsku.unit_cost_init with the unit cost from item_supp_country. If the cost_source on ordsku is 'MANL', only update if the deal_calc_queue.override_ind is 'Y'.

calculate_cost:

For this order , go through the ordsku_discount table and apply all off_invoice deals that should be applied. This means either all deals to be applied at time of order approval (deal_detail.deal_appl_timing = 'O')) for each sku on the order. (This is done regardless of whether the order is actually approved or not—if it is in worksheet status, we want to allow the user to see what the final costs would be). The base cost that discounts are calculated from should be ordsku.unit_cost_init for approved orders, and item_supp_country.unit_cost for unapproved orders. Update ordsku unit_cost to be this new cost (including discounts) and set cost_source to 'DEAL'. If elc_ind (from system_options) is 'Y', call apply_elc to calculate expenses and assessments for this order, supplier, and sku or pack.

If there is a pack number on ordsku_discount, use the unit_cost_init from ordsku where sku = pack number as the starting cost (for approved orders—use unit_cost from item_supp_country for the sku for unapproved orders). You must update unit cost on ordsku (for the pack's record) after each component sku is applied so that all the discounts will be included.

This data should all exist within the program and it should be possible to use it without going back and hitting the database again.

Business rules that need to be followed when applying discounts:

- Cumulative discounts need to be applied to the original unit cost (2% off + 3% off = 5 %off original unit cost)
- Cascade discounts need to be applied on the result thus far (take 2% off of the unit cost, then take 3% off of that price)
- for a % discount

discount cost = unit cost – (unit cost *%/100)

- for an amt discount (first convert amount to be in order currency if necessary)

discount cost = unit cost – amt (amount discounts are per unit cost already)

- fixed amt: if have fixed amt, must start with THAT amt rather than the unit cost (convert to order currency if necessary)

discount cost = fixed amt (converted to order currency if necessary)

- quantity discount (“buy some get some free”)

These are the most complicated. The cost of BOTH the free item AND the buy item will be affected. Unit costs need to be converted to order currency before any calculations are done, if necessary. If a buy/get free discount is encountered, the following things need to happen:

- Check whether the item is buy item or the free item (both will be on deal_itemloc). Only calculate the cost for whichever item you are actually dealing with right now; the other one will get calculated in its turn. If you are working on the buy item, you will already have its original unit cost and can skip the first part of the step below.
- If the buy item is not the same as the free item, get the original unit cost for the buy item from ordsku. If the buy item cost is less than the free item cost, stop; no discount will be taken.
- The unit cost used for the free item (before taking any discounts) should be deal_detail.qty_thresh_free_item_unit_cost.
- If the buy item is not on the order (fetching its original unit cost resulted in no data found), stop; no discount will be applied.
- If qty_ordered of the buy item < qty_thresh_buy_qty, stop; you didn’t get any discount
- Otherwise,
- Figure out how many free items you actually get.

- If the qty_thresh_recur_ind is 'N':

free qty = deal_detail.qty_thresh_free_qty

- If the qty_thresh_recur_ind is 'Y':

Buy item != free item:

free qty = floor(qty ordered of buy item / (qty_thresh_buy_qty)
* qty_thresh_free_qty

Buy item = free item:

free qty = floor(qty ordered / (qty_thresh_buy_qty + qty_thresh_free_qty))
* qty_thresh_free_qty

If the free item and the buy item are the same:

- Calculate the total discount:

total disc = free qty * qty_thresh_free_item_unit_cost

actual discount = total discount / qty ordered

- If the free item and the buy item are the same,

actual discount from above is the value that ordsku_discount.discount_value
will be updated to.

cost of free item with discount = qty_thresh_free_item_unit_cost – (total
disc / ordered qty) (this is what ordsku.unit_cost would get updated to)

- If the free item and the buy item are different:

- Calculate the total discount:

total discount = free qty * qty_thresh_free_unit_cost

- buy item total cost = buy qty_ordered * buy unit_cost

- total cost before discount = buy item total cost + total discount

buy item discount = buy item unit cost —

((buy item total cost / total cost before discount) *

buy item total cost) / buy item's qty. ordered)

free item discount =

((total discount / total cost before discount) * total discount) / free qty)

- This is the discount that should be subtracted from `ordsku.unit_cost` for each item and put in the `discount_value` on `ordsku_discount`. One record should be updated on `ordsku_discount` for each item
- Update the unit cost on `ordsku` (for the order number and the free and buy skus)

`apply_elc`:

Given an order number, sku, and pack (which might be NULL), and an order import indicator,

The “sku” below is the sku, vendor pack, or component sku from a buyer pack; the pack is the buyer pack id (or NULL if this is not a buyer pack).

Call `ELC_CALC_SQL.CALC_COMP` with an error message variable, a `calc_type` of ‘PE’, the sku as the item, the supplier, the pack, and the order number (all other parameters can be passed in as NULL). This will calculate expenses.

If `import_ind` is ‘Y’, call `ELC_CALC_SQL.CALC_COMP` as above, but with a `calc_type` of ‘PA’ to calculate assessments. Then call `ELC_CALC_SQL.CALC_COMP` with a `calc_type` of ‘PE’ to calculate any expenses that might have changed based on the assessments.

`update_OTB`:

For each item on the order that has an associated `otb_calc_type` (from the `deps` table) of ‘C’, call `OTB_SQL.ORD_RECEIVE`, passing in the difference between the original cost with ELC and the final cost with ELC (or just the difference between the original and final costs if ELC was turned off).

`delete_ordsku_disc_bld`:

Given an order number, delete all records on the `ordsku_discount_build` table that have that order number.

`size_arrays`:

Allocate memory for order and deal arrays.

`final`:

Restart/recovery close logic.

I/O Specification

N/A

Purchase Order Purge [ordprg]

Design Overview

The purpose of this module is to remove old orders from the system.

If the import indicator on the SYSTEM OPTIONS table (import_ind) is 'N' and if invoice matching is not installed, then all details associated with an order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months). If invoice matching is installed, then all details associated with an order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months). Orders are deleted only if shipments from the order have been completely matched to invoices or closed, and all those invoices have been posted.

If the import indicator on the SYSTEM OPTIONS table (import_ind) is 'Y' and if invoice matching is not installed, then all details associated with the order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months), as long as all ALC records associated with an order are in 'Processed' status, specified in ALC_HEAD (status). If invoice matching is installed, then all details associated with an order are deleted when the order has been closed for more months than specified in UNIT_OPTIONS (order_history_months), as long as all ALC records associated with an order are in 'Processed' status, specified in ALC_HEAD (status), and as long as all shipments from the order have been completely matched to invoices or closed, and all those invoices have been posted.

This program will also create a PO header flat file to interface with the Nautilus system. When orders are deleted, a record with the action type = 'D'leted will be written to an output file. Nautilus will then process this file and delete the PO from the warehouse's database to maintain consistency between the host and warehouse environment.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ALC_COMP_LOC	No	No	No	No	Yes
ALC_HEAD	No	Yes	No	No	Yes
ALLOC_DETAIL	No	No	No	No	Yes
ALLOC_HEADER	Yes	No	No	No	Yes
CE_LIC_VISA	No	No	No	No	Yes
CE_CHARGES	No	No	No	No	Yes
CE_ORD_ITEM	No	Yes	No	No	Yes
CE_SHIPMENT	No	No	No	No	Yes

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
CE_HEAD	No	No	No	No	Yes
INVC_DETAIL	Yes	No	No	No	Yes
INVC_HEAD	Yes	Yes	No	No	Yes
INVC_MATCH_WKSH T	Yes	No	No	No	Yes
INVC_MATCH_QUEUE	Yes	No	No	No	Yes
INVC_MERCH_VAT	Yes	No	No	No	Yes
INVC_NON_MERCH	Yes	No	No	No	Yes
INVC_XREF	Yes	No	No	No	Yes
OBLIGATION	No	No	No	No	Yes
OBLIGATION_COMP	No	No	No	No	Yes
OBLIGATION_COMP_ LOC	No	No	No	No	Yes
ORDCUST	No	No	No	No	Yes
ORDHEAD_DISCOUNT	No	No	No	No	Yes
ORDHEAD	Yes	Yes	No	No	Yes
ORDLOC	Yes	No	No	No	Yes
ORDSKU_INVC_COST	No	No	No	No	Yes
ORDSKU	Yes	No	No	No	Yes
LC_ORDAPPLY	No	No	No	No	Yes
DEAL_HEAD	No	No	No	No	Yes
DEAL_DETAIL	No	No	No	No	Yes
DEAL_ITEMLOC	No	No	No	No	Yes
DEAL_THRESHOLD	No	No	No	No	Yes
DEAL_QUEUE	No	No	No	No	Yes
DEAL_CALC_QUEUE	No	No	No	No	Yes
ORD_INV_MGMT	No	No	No	No	Yes
REPL_RESULTS	No	No	No	No	Yes
PERIOD	No	Yes	No	No	No
QC	Yes	No	No	No	Yes
RTV_DETAIL	No	No	No	No	Yes
SHIPMENT	Yes	Yes	No	No	Yes
SHPSKU	Yes	No	No	No	Yes

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
SYSTEM_OPTIONS	No	Yes	No	No	No
TRANS_CLAIMS	No	No	No	No	Yes
TRANS_DELIVERY	No	No	No	No	Yes
TRANS_LIC_VISA	No	No	No	No	Yes
TRANS_PACKING	No	No	No	No	Yes
TRANSPORTATION	No	Yes	No	No	Yes
TSFDETAIL	Yes	No	No	No	Yes
TSFHEAD	Yes	No	No	No	Yes
UNIT_OPTIONS	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE AD-HOC (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

N/A

Driving Cursor

When Retek's Invoice Matching product is not in use:

```
SELECT oh.order_no,  
       lc.lc_ind,  
       a.status  
FROM ordhead oh,  
       ordlc lc,  
       alc_head a  
WHERE lc.order_no(+) = oh.order_no
```



```

AND a.order_no(+) = oh.order_no

AND ((0 <
(NVL(MONTHS_BETWEEN(TO_DATE(:os_vdate,'YYYYMMDD'),
      oh.close_date),0) - :oi_hist_months))
or (oh.status = 'W'
AND oh.orig_ind = 0
AND oh.contract_no is NULL
AND (TO_DATE(:os_vdate,'YYYYMMDD') - oh.written_date) >=
:oi_repl_order_history_days));

```

When Retek's Invoice Matching product in use:

```

SELECT distinct oh.order_no,
      lc.lc_ind,
      a.status
FROM ordhead oh,
      shipment sh,
      shipsku ss,
      invc_head ih,
      ordlc lc,
      alc_head a
WHERE oh.order_no = sh.order_no
AND lc.order_no(+) = oh.order_no
AND a.order_no(+) = oh.order_no
AND sh.shipment = ss.shipment
AND ss.match_invc_id = ih.invc_id(+)
AND (0 <
(NVL(MONTHS_BETWEEN(TO_DATE(:os_vdate,'YYYYMMDD'),
      oh.close_date),0) - :oi_hist_months))

```

```
AND ((0 <
(NVL(MONTHS_BETWEEN(TO_DATE(:os_vdate,'YYYYMMDD'),
sh.invc_match_date),0) - :oi_hist_months))
OR (sh.invc_match_date IS NULL))
AND (ss.match_inv_id is null OR ih.status = 'P')
AND sh.invc_match_status = 'C'
AND 'C' = (SELECT decode(max(ship.invc_match_status),
min(ship.invc_match_status),'C','X')
FROM shipment ship
where ship.order_no = oh.order_no
group by ship.order_no)
UNION
SELECT oh.order_no,
lc.lc_ind,
a.status
FROM ordhead oh,
ordlc lc,
alc_head a
WHERE lc.order_no(+) = oh.order_no
AND a.order_no(+) = oh.order_no
AND oh.status = 'W'
AND oh.orig_ind = 0
AND oh.contract_no is NULL
AND (TO_DATE(:os_vdate,'YYYYMMDD') - oh.written_date) >=
:oi_repl_order_history_days;
```

Program Flow

N/A

Shared Modules

INV_SQL.DELETE_INV

Function Level Description

Delete from the appropriate ordering tables and any tables that may have referential integrity constraints for the fetched order number. Fetch order numbers appropriately based on whether or not invoice matching is installed.

Init()

Select the following fields values:

inv_match_ind, import_ind, repl_order_history_days, edi_rev_days, rws_ind
from the system_options table

order_history_months from the unit_options table

vdate from period table

Init_logistics()

Initialize the format of the output file.

Process()

Call del_rev to delete order revision

Open the particular driving cursor based on the indicator inv_match_ind

For each order:

Fetch the particular driving cursor based on inv_match_ind

If ordlc_lc_ind = 'Y' then skip and fetch next record;

If import_ind = 'Y' and alc_head.status = 'PR' then

Call delete_landed_costs;

End if;

Delete from related tables associated with orders being purged by.

Calling delete_inv_data, delete_deals and delete_repl_orders

Delete from ordhead table.

Delete_inv_data()

Call INVC_SQL.DELETE_INVC function to delete the invoice data for the specific orders being purged.

Del_rev()

Delete records from the tables ordloc_rev, ordsku_rev, ordhead_rev and alloc_rev associated with the orders which have been closed for more days than specified in edi_rev_days(in table UNIT_OPTIONS). But deleting occurs only:

when a letter of credit is not present(ordlc.lc_ind='N').

Import indicator equals 'N'. Or

import indicator equals 'Y', and the landed costs are completed (alc_head.status = 'PR'). In this case, purge these landed costs before deleting the above tables.

Also, before deleting from these tables, purge all related transportation and custom entries.

Purge_transport()

Delete from the table transportation for specific orders being purged as well as child records from tables missing_doc, trans_packing, trans_delivery, trans_claims and trans_lic_visa(based on transportation_id).

Purge_customs_entry()

Delete from customs entry for specific orders being purged.

Delete_landed_costs()

Delete landed costs and obligations as well as their child records for specific orders being purged. Involved tables include: alc_head, alc_comp_loc, obligation, obligation_comp, obligation_comp_loc.

Write_logistic_details()

Write the deleted order to the output file with action type = 'D'leted.

Delete_deals()

Delete all PO-specific deals assigned to the order being purged. PO-specific deals are identified by the existence of a value in deal_head.order_no.

Del_repl_orders ()

Delete records from the table ord_inv_mgmt and repl_results associated with the order being purged.

Final()

Close the output file.

I/O Specification

Output file: Format will be as in table given below. Each order is a separate transaction; multiple orders can be given in each file. The skeleton of file is:

- FHEAD (file identification – only one line per file) REQUIRED
- FDETL detail lines – one line for every record on ordhead. NOT REQUIRED.
- FTAIL (total number lines) One line per file. REQUIRED.

Record	Variable name	Field type	Default value	Description
FHEAD	File record descriptor	Char (5)	FHEAD	identifies file record type
	Line number	Number (10)		identifies file line number
	File type	Char (4)	POHD	Identifies file type
	Create date	Char (14)	Vdate	YYYYMMDDHH24MISS format
FDETL	File record descriptor	Char (5)	FHEAD	Identifies file record type
	Line number	Number (10)		Identifies file line number
	Action type	Char (1)	D	identifies the action type “D”elete
	Location	Char (4)		ordloc.location number
	File create date	Char (12)	vdate	YYYYMMDDHH24MI format
	Order number	Char (8)		ordhead.order_no
	Vendor	Char (7)		
	Preassigned Flag	Char (1)		
	Dev not before	Char (8)		
	Dev not after	Char (8)		
	Shipping terms	Char (3)		
	Buyer code	Char (12)		
FTAIL	File record descriptor	Char (5)	FTAIL	Identifies file record type
	Line number	Number (10)		Identifies file line number
	Transaction line number	Number (6)		identifies transaction’s line number

Technical Issues

N/A

Purchase Order Information Written to Order History Tables [ordrev]

Design Overview

Ordrev will write versions of approved order to order revision history tables. When orders are approved or when approved orders are modified, this program selects order numbers from the rev_orders table and writes current order information to the order/allocation revision tables. After the new version has been written to the order revision tables, all records will be deleted from the rev_orders table for that order_no.

This program processes order changes made by the client that may need to be sent to the vendor. The order changes should always be referred to as 'versions' and kept clearly distinct from order 'revisions' which are vendor changes uploaded via the ediupack program.

This program also allows Nautilus and Retek to interface, by sending the warehouse PO and allocation (ie. pre distribution) information to prepare the warehouse for incoming orders. The program will create two flat files, PO header and PO detail, based on approved orders found on the rev_orders table. The program will also create Pre Distribution Header and Pre Distribution Detail flat files, which will enable the warehouse to perform cross docking activities.

The last file produced by the ordrev batch program is a component ticketing file that holds retail and ticketing information for non sellable pack items. This file allows the warehouse to correctly ticket the components of the pack item, before distributing the items to the stores.

If an order is not in approved status at the time the batch program runs, then none of the above processing will occur. The record will stay on the rev_orders table until the PO is approved or deleted.

Scheduling Constraints

Processing Cycle: After rplprg & before edidlord, and Ad Hoc. This program must be run, if interfacing with Nautilus

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

Restartability will be implied, because the records that are selected from this table will be deleted before the commit. Restart library functions will still be included to ensure that rollback segments are not exceeded (by committing at intervals) and to perform basic record keeping functionality.

```
SELECT ro.action_type,
       ro.order_no,
       ro.alloc_no,
       ro.location,
       ro.sku,
       ro.hdr_dtl_ind,
       oh.pre_mark_ind,
       ro.rowid
FROM   rev_orders ro,
       ordhead oh
WHERE  ro.order_no = oh.order_no
       AND oh.status = 'A'
       AND MOD(ro.order_no, :oi_restart_num_threads) + 1 =
:oi_restart_thread_val
       AND ro.order_no > NVL(:ora_restart_order_no, -9999)
ORDER BY ro.order_no;
```

Program Flow

N/A

Shared Modules

PRICING_ATTRIB_SQL.GET_RETAIL(): get the unit retail from item_zone_pricing table for a sku/store.

PROMOTION_ATTRIB_SQL.EVENT_DESC(): get the event's description

Function Level Description

Init()

Initialization of the restart Retek recovery process will be performed here.

Get system date.

Open output files. There will be a maximum of 4 files (ie. one header and detail for PO download and one header and detail for Pre-distribution download)

Write FHEAD to all files.

Call Init_buffers().

Process()

All orders that need to have order version records will be processed.

If the order number changes, then perform the following logic.

The order number will be used to populate the revision history tables. The get_rev_no() function is called to determine the version number for the insert into the revision history tables.

Check if order is customer order. If order is customer order set flag to 1, else set to 0 (for the customer order no allocation information will be download to the RLS logistic).

If version 1 was just inserted (ie. order was just approved for the first time, no previous versions existed), then

Call write_new_po function to write newly created orders and associated allocations to the po header, po detail, pre distribution header, and pre distribution detail files.

Else

Call write_existing_po function to write the changed order information to the flat files. Some or all of the flat files may be written in this circumstance depending upon what information has changed since the order was last sent down to Nautilus.

End if;

The insert_header() function will be called from here to insert header level information, the insert_sku() function will insert order sku information, the insert_loc() will insert order sku/location information, and the insert_alloc() will insert order allocation information if the order's pre-mark indicator was set. This indicator will indicate whether cross-docked allocation information will be sent to the supplier along with the order. When all of the version information has been inserted into the revision history tables, all of the records with that order number should be deleted from the revord table by the delete_revord() function.

If system_options.financial_ap equals 'P', then call ins_revord () to insert into the fif_ordhead table.

Else /* the order number remains the same */

If order is not customer order. Call write_alloc_only().

End if;

Get_rev_no()

It is necessary to get the last version number that was written to the order revisions tables. The maximum version number is selected from the header revision table and then incremented by 1 to get the version number that will be inserted during processing. If no record exists in the order header revision history table, then the order is new and a version number of 1 is used.

Insert_header()

The current information on the order header table will be inserted into the header revision history table with the new version number

Insert_sku()

The current information on the order sku table is inserted into the order sku revision history table with the new version number

Insert_loc()

The current information on the order sku/location table is inserted into the order sku/location revision history table with the new version number.

Insert_alloc()

The ship-to warehouse on the allocation header table and the allocation information and quantity information from the allocation detail table is written to the allocation revision history table with the new version number.

Ins_revord()

Insert into the fif_ordhead table.

Write_new_po()

This function will write FDETL records to the appropriate PO and pre distribution output files.

Order information is retrieved from the ordhead and ordloc tables to populate the PO header and PO detail files. A record will be written to the PO download header and detail file for only orders with a warehouse destination. The warehouse number will be stored in the Location (DC) field on the file. If the order is going to other locations, such as stores, then do not write a record to the files. There will be one header for each order/wh location retrieved.

Check customer order flag. If it is not customer order, open a “for loop” to retrieve the allocation information for an order.

Write pre-distribution header and detail with action type = ‘A’dd for the warehouse/allocation/sku/order_no. There will be one header for every alloc_no retrieved and a detail record for each to_location for that allocation. In other words, the first allocation number will be written to the pre-distribution header record. Write the pre-distribution detail records, until that allocation number changes. When the allocation number changes, then write a pre-distribution header record. The warehouse (from_loc) will be stored in the Location (DC) field on the file. Call promotion_attrb_sql.get_event_desc package for the event’s description. Also, get the correct retail (pricing_attrb_sql.get_retail package) and ticketing information for the predistribution detail file. In the for loop, if the allocation location is a store, call comp_tckt () function to write the component ticketing file.

Write_existing_po()

Open a “for loop” to retrieve ordhead and ordloc fields for comparison. The comparison will be completed for each warehouse location the order is destined. In the for loop, compare ordhead/ordloc with previous version on ordhead_rev/ordloc_rev. If there are any changes to the Nautilus required fields, then write PO download header and/or detail records. This process only needs to be done for orders going to warehouse locations.

Fetch the header information from ordhead and ordhead_rev. Compare each field (ie. ordhead.buyer = ohr.buyer). If the fields do not equal, then set an indicator, which will indicate that the ordhead records have been modified and an action_type = ‘M’ will need to be sent down in the PO header file.

For the order number retrieved in the above cursor loop through the ordloc warehouse records. First, check the header indicator. If the ordhead record has changed, then a PO header record needs to be written for each warehouse on the order. For example, one PO (#123456) has been created to replenish the stock in warehouse 1, 2, and 3. The PO header download file produced by the ordrev.pc program will have 3 separate records. The first FDETL will have a location (DC) = 1 for PO #123456, the second record will have a location (DC) = 2 for PO #123456, and the third record will have a location (DC) = 3 for PO#123456. After the ordhead indicator check, compare the ordloc and the ordloc_rev fields. If one of the fields differ, then write a PO detail record for the warehouse/order_no. Once all warehouse locations are processed in that order, go fetch the next order.

If ordloc.qty_ordered != 0, then action type = 'M'

If ordloc.qty_ordered = 0, then action type = 'D'

Check customer order flag. If it is not customer order. Call write_alloc_only();

Write_alloc_only()

This function will write FDETL records to the appropriate pre-distribution output files.

If alloc_no is not NULL, then (alloc_no was retrieved from the main driving cursor on the rev_orders table)

If location is NULL and action type = 'A' then

Write pre-distribution download header and detail with action type 'A'. If the action type = 'A', then loop through all of the "to locations" of the allocation on alloc_detail table. A detail record will need to be written for each alloc_detail location.

In the for loop, if the allocation location is a store, then call the comp_tckt() function.

Elsif location is not NULL and action type = 'D' and hdr_dtl_ind = 'H'

Write pre-distribution header with action type = 'D'. The location field retrieved by the driving cursor will contain the from warehouse location (ie. alloc_header.wh) and should be used to populate the Location (DC) field on the output file.

Elsif location is not NULL and action type = 'D' and hdr_dtl_ind = 'D'

Write pre-distribution detail with action type = 'D'. The location field on the rev_orders table will contain the to store/warehouse location (ie. alloc_detail.store or wh) and should be used to populate the destination id on the output file.

Else /* location is not NULL and action type = 'A' or 'M' */

Write pre-distribution download detail with 'A', 'M', depending on the action type retrieved from the main cursor (ie. rev_orders). Get the detail file's information (from_loc, to_loc, qty) by selecting from the alloc_detail/alloc_header table for the alloc_no and location found in the main driving cursor. A detail record should be written for the location that was retrieved from the rev_orders table.

If the action type = 'A' and the allocation location is a store, then call the comp_tckt() function.

End if;

End if;

Comp_tckt()

If the sku on the allocation is a non sellable pack item going to a store location, then write all of the component skus, retail price, and ticket information to the component_ticketing file.

Del_revord()

Multiple order versions could exist on the revord table for the same order. This could happen if the batch program had not been run since the last time the order was modified. Since the processing has written the current order value to the revision history tables, all records with that order number must be deleted from the revord table to prevent double processing

I/O Specification

The five output files should be specified at the command line when running the ordrev.pc program.

Order Header file

Record Name	Record	Default value	Field type	Description
File Header	Detail file identifier	FHEAD	Char(5)	Identifies the header line
	line number	Incremented internally	Number(10)	sequential line number
	Program descriptor	POHD	Char(5)	Identifies the program
	Create date	YYYYMMDDHH 24MISS	Char(14)	File create date

Record Name	Record	Default value	Field type	Description
File detail	File record descriptor	FDETL	Char(5)	Identifies the detail line
	Line number	Incremented internally	Number(10)	sequential line number
	Action_type	'A', 'M', 'D'	Char(1)	Add, modify, or delete action type
	Location	Ordloc.location (wh only)	Number(4)	Location of item that was ordered
	Transaction day date/time	sysdate	Datetime(12)	system date
	Po number	'P' + ordhead.order_no	Char(9)	Unique identifier of the purchase order, prefixed with 'P'
	Vendor number	Ordhead.supplier	Number(7)	Supplier number of the order
	Preassigned flag	'N'	Char(1)	
	Deliver_not_before_date	Not_before_date	Date(8)	Not_before_date of the order
	Deliver_not_after_date	Not_after_date	Date(8)	Not_after_date of the order
	Shipping terms	Ordhead.freight_terms	Char(3)	Freight Terms of the order
	Buyer code	Ordhead.buyer	Char(12)	Buyer of the PO.
File trailer	File record identification	FTAIL	Char(5)	File trailer identifier
	Line number	Internally incremented	Number(10)	Sequential line number of file
	Number of transaction lines	Internally determined	Number(10)	Total number of transactions (not including FHEAD and FTAIL)

Order Detail file

Record Name	Record	Default value	Field type	Description
File header	File line identifier	FHEAD	Char(5)	Identifies file record type
	Line number	Begins at 0000000001	Number(10)	Identifies file line number
	Program descriptor	PODT	Char(5)	identifies the program
	Create date	YYYYMMDDHH24MISS format	Char(14)	file create date
File Detail	Detail file identifier	FDETL	Char(5)	Identifies the Detail line
	line number	Incremented internally	Number(10)	sequential line number
	Action_type	'A', 'M', 'D'	Char(1)	Add, modify, or delete action type
	Location	Ordloc.location (wh only)	Number(4)	This field contains the location to which the item will be ordered to.
	Transaction day date/time	sysdate	Datetime(12)	system date
	PO number	'P' + order number	char(9)	Identifies the unique PO number
	Item id	Ordloc.sku	Char(16)	Sku on the order
	Requested unit qty	Ordloc.qty_ordered	Number(12,4)	Contains the total number of items ordered to a specific location.
	Ordered case pack	Ordsku.case_pack_size	Number(12,4)	Contains the case pack size that the item was ordered in
	Hang/Flat/Shoe Indicator	Hanger attribute or default door type	Char(1)	F=Flat, H=Hang, S=Shoe, A=All

Record Name	Record	Default value	Field type	Description
File Trailer	File Line identifier	FTAIL	Char(5)	Identifies the trailer line
	line number	Incremented internally	Number(10)	sequential line number
	number of transaction lines	Total number of detail lines	Number(10)	total number of detail lines in file (not including FHEAD and FTAIL)

Stock Order file

Record Name	Record	Default value	Field type	Description
File Header	Detail file identifier	FHEAD	Char(5)	Identifies the header line
	line number	Incremented internally	Number(10)	sequential line number
	Program descriptor	STOR	Char(5)	Identifies the program
	Create date	YYYYMMDDHH 24MISS	Char(14)	File create date
File detail	File record descriptor	FDETL	Char(5)	Identifies the detail line
	Line number	Incremented internally	Number(10)	sequential line number
	Action_type	'A', 'M', 'D'	Char(1)	Add, modify, or delete action type
	Location	alloc_header.wh	Number(4)	From Warehouse location
	Transaction day date/time	sysdate	Datetime(12)	system date
	distribution number	'A' + alloc_no	char(9)	Allocation number. Prefix 'A' for alloc

Record Name	Record	Default value	Field type	Description
	Download comment	NULL	Char(30)	Comment to be printed on the label (for future use)
	Pick_not_before_date	Not_before_date	Date(8)	Not_before_date of the order
	Pick_not_after_date	Not_after_date	Date(8)	Not_after_date of the order
	Event code	Promotion or NULL	Char(6)	Promotion's event number
	Event description	Prom_desc or NULL	Char(25)	Event description
	priority	1	Char(4)	Priority
	Order Type	ALLOC_HEADER.ORDER_TYPE	Char(9)	Type of Order : 'PO' or 'PREDIST'
	Break by Distro	'N'	Char(1)	Controls the mixing of orders (distros) in a container
	Carrier Code	NULL	Char(4)	Code of the carrier for the order
	Carrier Service Code	NULL	Char(6)	Carrier's service code for the delivery, First Class, and so on (Future Use)
	Route	NULL	Char(10)	Route specified for the delivery
	Ship Address Description	NULL	Char(30)	The description (such as the store name)
	Ship Address Line 1	NULL	Char(30)	Shipping Address Line 1

Record Name	Record	Default value	Field type	Description
	Ship Address Line 2	NULL	Char(30)	Shipping Address Line 2
	City	NULL	Char(25)	Shipping City
	State	NULL	Char(3)	Shipping State
	Zip	NULL	Char(10)	Shipping Zip
	Billing Address Description	NULL	Char(30)	The description (such as company name). This is the first line of the address block.
	Billing Address 1	NULL	Char(30)	Billing Address Line 1
	Billing Address 2	NULL	Char(30)	Billing Address Line 2
	Billing Address 3	NULL	Char(30)	Billing Address Line 3
	Amount 1	NULL	Number(8,2)	Amount Charge 1
	Amount 2	NULL	Number(8,2)	Amount Charge 2
	Amount 3	NULL	Number(8,2)	Amount Charge 3
	PO Number	'P' + ALLOC_HEADER .ORDER_NO	Char(9)	Unique identifier of the purchase order, prefixed with 'P'.
File trailer	File record identification	TTAIL	Char(5)	File trailer identifier
	Line number	Internally incremented	Number(10)	Sequential line number of file

Record Name	Record	Default value	Field type	Description
	Number of transaction lines	Internally determined	Number(6)	Total number of transactions (not including FHEAD and FTAIL)

Stock Allocation file

Record Name	Record	Default value	Field type	Description
File header	File line identifier	FHEAD	Char(5)	identifies file record type
	Line number	Begins at 0000000001	Number(10)	identifies file line number
	Program descriptor	STAL	Char(10)	identifies the program
	Create date	YYYYMMDDHH24 MISS format	Char(14)	file create date
File Detail	Detail file identifier	FDETL	Char(5)	Identifies the Detail line
	line number	Incremented internally	Number(10)	sequential line number
	Action_type	'A', 'M', 'D'	Char(1)	Add, modify, or delete action type
	Location	alloc_header.wh	Number(4)	From Warehouse location
	Transaction day date/time	Sysdate	Datetime(12)	system date
	distribution number	'A' + alloc_no	char(9)	Allocation number. Prefix 'A' for alloc
	Item Id	ALLOC_HEADER.SKU	Char(16)	Unique item identifier
	requested unit qty	Alloc_detail.qty_allocated	Number(12,4)	quantity allocated
	destination id	Alloc_detail.store or wh	Number(4)	Allocation location

Record Name	Record	Default value	Field type	Description
	price	Item_zone_price.unit_retail	Number(5,2)	Retail price
	print upc flag	NULL	char(1)	Print upc flag
	ticket type	item_ticket.ticket_type	Number(4)	Receiving Ticket type of item.
	priority	1	Char(4)	Priority
	expedite flag	'N'	char(1)	Flag indicating whether the order should be shipped via normal or expedite carrier service.
File Trailer	File Line identifier	FTAIL	Char(5)	Identifies the trailer line
	line number	Incremented internally	Number(10)	sequential line number
	number of transaction lines	Total number of detail lines	Number(6)	total number of detail lines in file (not including FHEAD and FTAIL)

Component Ticketing file layout:

Record Name	Record	Default value	Field type	Description
File Header	File Line identifier	FHEAD	Char(5)	Identifies the trailer line
	Line number	0000000001	Number(10)	identifies file line number
	Program descriptor	CPTT	Char(4)	identifies the program
	Create date	YYYYMMDDHH24MISS	Char(14)	file create date

Record Name	Record	Default value	Field type	Description
File detail	file record descriptor	FDETL	Char(5)	Detail line descriptor
	line number	Incremented internally	Number(10)	sequential line number
	Action_type	'A'	Char(1)	'A'dd, 'M'odify, 'D'elele
	Location	alloc_header.wh	Number(4)	location that items will be allocated from
	Transaction date/time	vdate	Datetime(12)	date/time created in RMS
	distribution number	alloc_header.alloc_no	char(9)	Unique identifier of the distribution.
	Master item id	alloc_header.sku	Char(16)	Unique identifier of the pack item
	Dest Id	alloc_detail.store	Number(4)	Identifier of the ship destination
	Component Item ID	v_packsku_qty.sku	Char (16)	item identifier of the component
	price	Item_zone_price.unit_retail	Number(7,2)	Price of the merchandise.
File Trailer	file record identification	FTAIL	Char(5)	File trailer
	line number	Incremented internally	Number(10)	sequential line number
	number of transaction lines	Total number of detail lines	Number(6)	total number of transaction lines in file (not including FHEAD and FTAIL)

Retail Price Change on Purchase Orders [ordupd]

Design Overview

This program will be used to automatically change all retail costs on purchase orders when a retail price change is implemented for an item on the order with the status of 'Worksheet', 'Submit' and 'Approve'.

Open To Buy is updated to give a more accurate picture of the retail value of open orders if the order is 'Approved' and if the department calculate the OTB as Retail.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DEPS	No	Yes	No	No	No
DISPLAY_PACKITEM	No	Yes	No	No	No
ORDDEPT	No	Yes	No	No	No
ORDHEAD	No	Yes	No	No	No
ORDLOC	No	Yes	No	Yes	No
OTB	No	Yes	No	Yes	No
PERIOD	No	Yes	No	No	No
PRICE_HIST	No	Yes	No	No	No
V_SKU_INFO	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE 4 (daily)

Scheduling Diagram: This program should be run after pccxt.pc and pccxt.pc to ensure that all price changes have been handled by batch processing.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: This module does not contain restart/recovery.

Restart Recovery

This module does not contain restart/recovery.

```
EXEC SQL DECLARE c_affected_orders CURSOR FOR
select /*+ INDEX(PH,PRICE_HIST_I2)
      INDEX(OL,ORDLOC_I1)
      INDEX(OH,PK_ORDHEAD)
      */
distinct 'S',
oh.order_no,
oh.currency_code,
0 pack_no,
TO_CHAR(oh.otb_eow_date, 'YYYYMMDD') eow_date,
oh.order_type,
oh.status,
ph.unit_retail - NVL(ol.unit_retail, 0) ordloc_retail,
NVL(ol.qty_ordered, 0) - NVL(ol.qty_received, 0) qty_outstanding,
ph.sku,
ol.loc_type,
ol.location,
0 pack_qty,
ph.unit_retail price_hist_unit_retail
from price_hist ph,
      ordloc ol,
      ordhead oh
where oh.order_no = ol.order_no
      and ph.sku = ol.sku
```

```
and (ph.store      = ol.location
or  ph.wh         = ol.location)
and ph.tran_type  in (4, 8, 11)
and ph.action_date = TO_DATE(:os_tomorrow, 'YYYYMMDD')
and oh.status     in ('W', 'S', 'A')
UNION ALL
select /*+ INDEX(PH,PRICE_HIST_I2)
      INDEX(OL,ORDLOC_I1)
      INDEX(OH,PK_ORDHEAD)
      */
distinct 'P',
oh.order_no,
oh.currency_code,
vpq.pack_no,
TO_CHAR(oh.otb_eow_date, 'YYYYMMDD') eow_date,
oh.order_type,
oh.status,
NVL(ol.unit_retail, 0) ordloc_retail,
vpq.qty * (NVL(ol.qty_ordered, 0) - NVL(ol.qty_received, 0))
  qty_outstanding,
ph.sku,
ol.loc_type,
ol.location,
vpq.qty pack_qty,
ph.unit_retail price_hist_unit_retail
from ordloc      ol,
```



```
ordhead    oh,  
v_packsku_qty vpq,  
price_hist ph  
where ol.sku      = vpq.pack_no  
and ph.sku       = vpq.sku  
and oh.order_no  = ol.order_no  
and oh.status    in ('W', 'S', 'A')  
and (ph.store    = ol.location  
or ph.wh        = ol.location)  
and ph.tran_type in (4, 8, 11)  
and ph.action_date = TO_DATE(:os_tomorrow, 'YYYYMMDD');
```

Program Flow

N/A

Shared Modules

CURRENCY_SQL.CONVERT_BY_LOCATION - use this package for all conversions.

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Outstanding Order Export Download File [otbdlord]

Design Overview

This batch program will sum outstanding orders from past periods for each subclass and export the data to a flat file for use by an outside planning system.

Approved orders of each order type are summed by subclass across all periods with an end-of-week date before today. The outstanding order amount is then calculated by subtracting all receipts for each order type by subclass from the approved order amount. This figure is written to the output file for each order type by subclass.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
OTB	Yes	Yes	No	No	No
PERIOD	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: At user discretion.

Scheduling Diagram: This program must be run after salweek for the week just ended. This program and otbdlsal can run anytime after salweek, but saldly cannot run between otbdnld, otbdlsal and otbdlord

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Since array fetches are used in this program, only one thread will be used.

Restart Recovery

The logical unit of work for the otbdlord module is department/class/subclass with a recommended commit counter setting of 10,000. Each time the record counter equals the maximum recommended commit number, an application image array record will be written to the restart_start_array for restart/recovery if a fatal error occurs.

```

SELECT      dept,

class,

subclass,

ROUND(SUM(NVL(n_approved_amt,0)) -

        SUM(NVL(n_receipts_amt,0)),4),

ROUND(SUM(NVL(b_approved_amt,0)) -

        SUM(NVL(b_receipts_amt,0)),4),

ROUND(SUM(NVL(a_approved_amt,0)) -

        SUM(NVL(a_receipts_amt,0)),4)

FROM otb

WHERE      eow_date <

TO_DATE(file_create_date,'YYYYMMDD')

AND  (dept > NVL(:ora_restart_dept, -999) or

(dept = :ora_restart_dept and

        (class > :ora_restart_class or

(class = :ora_restart_class and

subclass > :ora_restart_subclass)

)

)

)

GROUP BY  dept,

class,
```

```
        subclass
ORDER BY    dept,
           class,
           subclass;
```

Program Flow

declare the start string array to include department, class and subclass

declare output file variables

```
main()
{
    accept the output file name as an argument from the command line
    init()
    {
        initialize restart/recovery
        assign restart variables
        copy start string variables from the restart_start_array

        fetch today's date and, if not restarting, select
otb_file_rec_sequence.NEXTVAL

        open the temp output file

        if not restarting, write a header record to the temp file
    }

    declare the driving cursor

    table_process()
    {
        set up out_buffer to receive fetched data
```

```
        open the driving cursor
    LOOP
        fetch
        if no rows are found, set the end flag
        break if there are no rows left to process
    process_records()
        LOOP through the records to be processed
        increment the file line sequence number
        print each record fetched to the temp file
    end LOOP
    break if the end flag is set
    set the application image array and call restart_commit()
to save it
        call restart_file_write() to write successfully processed
records from the temp file to the final output file
    end LOOP
    close the driving cursor
    write a trailer record to the temp file
}

final()
{
    close restart/recovery logic
    if no errors were encountered in the last round of processing,
write any remaining records in the temp file to the final output file
    close the export file
    remove the temp file
}
```

Shared Modules

N/A

Function Level Description

Restart/recovery for table-based processing should be implemented for this program.

main():

Accept three runtime parameters at the command line: program name, user name/password, and output file name.

init():

Initialize restart/recovery.

Open the temp file for output.

Fetch today's date into file_write_date.

If not restarting, fetch otb_file_rec_sequence.NEXTVAL into transaction_set_control_number and write the file header record to the temp file.

Set up an array to receive fetches in table_process() - memory should be dynamically allocated so that the cursor fetches arrays of size g_l_restart_max_counter.

table_process():

Set up out_buffer to receive fetched data:

```
strcpy(out_buffer, "%5s"); /* file type record descriptor */  
strcat(out_buffer, "%-10d"); /* file line sequence number */  
strcat(out_buffer, "%-14d"); /* transaction set control number */  
strcat(out_buffer, "%04d"); /* dept */  
strcat(out_buffer, "%04d"); /* class */  
strcat(out_buffer, "%020.0f"); /* a_outs_amt */
```

Each strcpy/strcat combination should set up space in out_buffer for a single fetched variable. It should copy the appropriate data type (d for decimal integers or s for strings), and all numbers should be left-padded with zeroes except counters, which are left-justified and right-padded with spaces. The length of the space should correspond with the variable's length on the OTB table (e.g. cancel_amt is an oracle variable of (20,4)--it needs a space 20 wide and should be left-padded with zeroes (%020.0f)). The name of the variable that will be copied into that space is included as a comment to ensure maintainability of this code.

Open the driving cursor C_OUTS_ORD.

Use a while loop to fetch rows from the table into the arrays.

If no rows are found on the table, set a flag to indicate that processing should end after this array.

Set file_type_record_descriptor to 'FDETL'.

Break if there are no more rows to process.

Call process_records() to process the array.

Break if the end flag has been set.

Copy the last row's start string variables into ora_restart_cur_string, delimited by semicolons. This will serve as the application image array. Call the restart_commit() function to save it.

Call the restart_file_write() function to write successfully processed records to the temp file.

End while loop.

Close C_OUTS_ORD.

Write a trailer record to the temp file with control_number_file_line_count = file_line_sequence_number - 2 and file_type_record_descriptor = 'FTAIL'.

final():

Close restart/recovery logic.

If no errors were encountered in the final round of processing, write any last records from the temp file to the final output file.

Close the output file and delete the temp file.

process_records():

Loop through a subset of records and write each of them to the temp file using out_buffer as a format mask.

The file_line_sequence_number should be incremented each time a record is written.

I/O Specification

Output File:

The output file name should be accepted as a runtime parameter at the command line.

The file will be formatted as fixed-width values with no delimiters or punctuation. A new-line character will indicate the end of each record.

A sample detail record would look like the following:

```
FDETL1123456789012341234123411111111111111111111222222222222  
222222222333333333333333333333333333
```

A sequence will need to be added to the database to increment the Transaction Set Control Number. This sequence should be large enough that there is very little danger of running out of numbers. It will give a unique identifying number to every file exported from Retek to an outside planning system.

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	Keeps track of the record's position in the file by line number
	File Type Definition	Char(4)	OTBE	Identifies file as 'OTB Export'
	File Create Date	Date	vdate	Date file was written by batch program
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	Keeps track of the record's position in the file by line number
	Transaction Set Control Number	Number(14)	sequence number	Used to force unique file check
	Department	Number(4)	dept identifier	The id number of a department

Record Name	Field Name	Field Type	Default Value	Description
	Class	Number(4)	class identifier	The id number of a class within the department given
	Subclass	Number(4)	subclass identifier	The id number of a subclass within the class given
	N Outstanding Amt	Number(20, 4)	non-basic outstanding order amt	The amt. of outstanding non-basic orders for past periods
	B Outstanding Amt	Number(20, 4)	buyer-basic outstanding order amt	The amt. of outstanding buyer-basic orders for past periods
	A Outstanding Amt	Number(20, 4)	auto-basic outstanding order amt	The amt. of outstanding auto-basic orders for past periods
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	Keeps track of the record's position in the file by line number
	Control Number File Line Count	Number(10)	total detail lines	sum of all transaction lines, not including file header and trailer

Error File

Standard Retek batch error handling will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Stock Ledger Data Download [otbdlsal]

Design Overview

This module will sum stock ledger data from the DAILY_DATA table and opening stock information from the WEEK_DATA table across the current week, grouping by department, class, subclass, location and date, and export the data to a flat file for use by an outside planning system.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DAILY_DATA	Yes	Yes	No	No	No
WEEK_DATA	Yes	Yes	No	No	No
PERIOD	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE 4 (weekly/monthly)

Scheduling Diagram: This program must be run after salweek for the week just ended. This program and otbdnld can run anytime after salweek, but saldly cannot run between otbdnld, otbdlsal and otbdlord.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Since array fetches are used in this program, only one thread will be used.

Restart Recovery

The logical unit of work for the otbdlsal module is department, class, subclass, store, warehouse, half, month and week with a recommended commit counter setting of 10,000. Each time the record counter equals the maximum recommended commit number, an application image array record will be written to the restart_start_array for restart/recovery if a fatal error occurs.

```
SELECT      w.dept,

w.class,

w.subclass,

w.store,

w.wh,

w.half_no,

w.month_no,

w.week_no,

NVL(w.opn_stk_retail,0),

NVL(w.opn_stk_cost,0),

ROUND(SUM(NVL(d.stock_adj_retail,0)),4),

ROUND(SUM(NVL(d.stock_adj_cost,0)),4),

ROUND(SUM(NVL(d.purch_retail,0)),4),

ROUND(SUM(NVL(d.purch_cost,0)),4),

ROUND(SUM(NVL(d.rtv_retail,0)),4),

ROUND(SUM(NVL(d.rtv_cost,0)),4),

ROUND(SUM(NVL(d.freight_cost,0)),4),

ROUND(SUM(NVL(d.net_sales_retail,0)),4),

ROUND(SUM(NVL(d.net_sales_cost,0)),4),

ROUND(SUM(NVL(d.returns_retail,0)),4),

ROUND(SUM(NVL(d.returns_cost,0)),4),

ROUND(SUM(NVL(d.prom_markdown_retail,0)),4),
```

```
ROUND(SUM(NVL(d.markdown_can_retail,0)),4),
ROUND(SUM(NVL(d.empl_disc_retail,0)),4),
ROUND(SUM(NVL(d.workroom_amt,0)),4),
ROUND(SUM(NVL(d.cash_disc_amt,0)),4),
ROUND(SUM(NVL(d.sales_units,0)),4),
ROUND(SUM(NVL(d.markup_retail,0)),4),
ROUND(SUM(NVL(d.markup_can_retail,0)),4),
ROUND(SUM(NVL(d.clear_markdown_retail,0)),4),
ROUND(SUM(NVL(d.perm_markdown_retail,0)),4)
FROM week_data w, daily_data d
WHERE      w.dept = d.dept
AND    w.class = d.class
AND    w.subclass = d.subclass
AND    w.store = d.store
AND    w.wh = d.wh
AND    w.eow_date = L_eow_date
AND    d.data_date >= L_eow_date - 6
AND    (w.dept > NVL(:ora_restart_dept, -999) or
(w.dept = :ora_restart_dept and
(w.class > :ora_restart_class or
(w.class = :ora_restart_class and
(w.subclass > :ora_restart_subclass or
(w.subclass = :ora_restart_subclass and
(w.store > :ora_restart_store or
(w.store = :ora_restart_store and
w.wh > :ora_restart_wh)))))))))
```

```
GROUP BY  w.dept,
          w.class,
          w.subclass,
          w.store,
          w.wh,
          w.half_no,
          w.month_no,
          w.week_no,
          w.opn_stk_retail,
          w.opn_stk_cost
ORDER BY  w.dept,
          w.class,
          w.subclass,
          w.store,
          w.wh;
```

Program Flow

declare the start string array to include department, class, subclass, store, warehouse

declare output file variables

main()

accept the output file name as an argument from the command line

init()

{

initialize restart/recovery

```
        assign restart variables

        copy start string variables from the restart_start_array

        fetch today's date and, if not restarting, select
otb_file_rec_sequence.NEXTVAL

        open the temp output file

        if not restarting, write a header record to the temp file

    }

declare the driving cursor

table_process()

{

    call CAL_TO_454_LDOW to get the last day of the current
week

    set up out_buffer to receive fetched data

    open the driving cursor

    LOOP

        fetch

        if no rows are found, set the end flag

        break if there are no rows left to process

        process_records()

            LOOP through the records to be processed

                increment file_line_sequence_number

                print each record fetched to the temp file

            end LOOP

        break if the end flag is set

        set the application image array and call restart_commit()
to save it
```

call restart_file_write() to write successfully processed records from the temp file to the final output file

end LOOP

close the driving cursor

write a trailer record to the temp file

}

final()

{

close restart/recovery logic

if no errors were encountered in the last round of processing,
write any remaining records in the temp file to the final output file

close the export file

remove the temp file

}

Shared Modules

CAL_TO_454_LDOW to get the end-of-week date for the current week.

Function Level Description

Restart/recovery for table-based processing should be implemented for this program. Use the restart_skeleton program as a framework.

main():

- Accept three runtime parameters at the command line: program name, user name/password, and output file name.

init():

- Initialize restart/recovery.
- Open the temp file for output.
- Fetch today's date into file_write_date.
- If not restarting, fetch otb_file_rec_sequence.NEXTVAL into transaction_set_control_number and write a file header record to the temp file.
- Set up an array to receive fetches in table_process() - memory should be dynamically allocated so that the cursor fetches arrays of size g_l_restart_max_counter.

table_process():

- Pass file_write_date into CAL_TO_454_LDOW to get the last day of the current week.
- Set up out_buffer to receive fetched data:

```
strcpy(out_buffer, "%5s"); /* file type record descriptor */  
strcat(out_buffer, "%-10d"); /* file line sequence number */  
strcat(out_buffer, "%-14d"); /* transaction set control number */  
strcat(out_buffer, "%04d"); /* dept */  
strcat(out_buffer, "%04d"); /* class */  
....  
strcat(out_buffer, "%020.0f"); /* opn_stk_retail */  
....
```

Each strcpy/strcat combination should set up space in out_buffer for a single fetched variable. It should copy the appropriate datatype (d for decimal integers or s for strings), and all numbers should be left-padded with zeroes except counters, which are left-justified and right-padded with spaces. The length of the space should correspond with the variable's length on the OTB table (e.g. cancel_amt is an oracle variable of (20,4)--it needs a space 20 wide and should be left-padded with zeroes (%020d)). The name of the variable that will be copied into that space is included as a comment to ensure maintainability of this code.

- Open the driving cursor C_STK.
- Use a while loop to fetch rows from the table into the arrays.
- If no rows are fetched from the table, set a flag to indicate that processing should end after this array.
- Set file_type_record_descriptor to 'FDETL'.
- Break if there are no more rows to process.
- Call process_records() to process the array.
- Break if the end flag has been set.
- Copy the last row's start string variables into ora_restart_cur_string, delimited by semicolons. This will serve as the application image array. Call the restart_commit() function to save it.
- Call the restart_file_write() function to write successfully processed records to the temp file.
- End while loop.
- Close C_STK.
- Write a trailer record to the temp file with control_number_file_line_count = file_line_sequence_number - 2 and file_type_record_descriptor = 'FTAIL'.

final():

- Close restart/recovery logic.
- If no errors were encountered in the final round of processing, write any remaining records from the temp file to the final output file
- Close the output file and delete the temp file.

process_records():

- Loop through a subset of records and write each of them to the temp file using out_buffer as a format mask.
- The file_line_sequence_number should be incremented each time a record is written.

I/O Specification

Record Name	Field Name	Field Type	Default Value	Description
	Class	Number(4)	class identifier	The id number of a class within the department given
	Subclass	Number(4)	subclass identifier	The id number of a subclass within the class given
	Store	Number(4)	store identifier	The store from which stock ledger data was collected (-1 if this location is a WH)
	Wh	Number(4)	warehouse identifier	The wh from which stock ledger data was collected (-1 if this location is a store)
	Half No.	Number(5)	half no.	The half number for this stock ledger data
	Month No.	Number(2)	month no.	The month number in the half for this stock ledger data
	Week No.	Number(2)	week no.	The week number in the month for this stock ledger data
	Open Stock Retail	Number(20,4)	opn stk retail	The retail opening stock from the week_data table for this stock ledger period
	Open Stock Cost	Number(20,4)	opn stk cost	The cost opening stock from the week_data table for this stock ledger period
	Stock Adjustments Retail	Number(20,4)	stock adj retail	The retail stock adjustments summed from the daily_data table for this stock ledger period
	Stock Adjustments Cost	Number(20,4)	stock adj cost	The cost stock adjustments summed from the daily_data table for this stock ledger period
	Purchases Retail	Number(20,4)	purch retail	The retail purchases summed from the

Record Name	Field Name	Field Type	Default Value	Description
				daily_data table for this stock ledger period
	Purchases Cost	Number(20,4)	purch cost	The cost purchases summed from the daily_data table for this stock ledger period
	RTV Retail	Number(20,4)	rtv retail	The retail return to vendor amount summed from the daily_data table for this stock ledger period
	RTV Cost	Number(20,4)	rtv cost	The cost return to vendor amount summed from the daily_data table for this stock ledger period
	Freight Cost	Number(20,4)	freight cost	The freight cost summed from the daily_data table for this stock ledger period
	Net Sales Retail	Number(20,4)	net sales retail	The retail net sales summed from the daily_data table for this stock ledger period
	Net Sales Cost	Number(20,4)	net sales cost	The cost net sales summed from the daily_data table for this stock ledger period
	Returns Retail	Number(20,4)	returns retail	The retail returns amount summed from the daily_data table for this stock ledger period
	Returns Cost	Number(20,4)	returns cost	The cost returns amount summed from the daily_data table for this stock ledger period
	Promotional Markdowns Retail	Number(20,4)	prom markdown retail	The retail promotional markdowns summed from the daily_data table for this stock ledger period

Record Name	Field Name	Field Type	Default Value	Description
	Markdown Cancellations Retail	Number(20,4)	markdown can retail	The retail markdown cancellations summed from the daily_data table for this stock ledger period
	Employee Discount Retail	Number(20,4)	empl disc retail	The retail employee discounts amount summed from the daily_data table for this stock ledger period
	Workroom Amount	Number(20,4)	workroom amt	The workroom amount summed from the daily_data table for this stock ledger period
	Cash Discount Amount	Number(20,4)	cash disc amt	The cash discounts amount summed from the daily_data table for this stock ledger period
	Sales Units	Number(12,4)	sales units	The sales units summed from the daily_data table for this stock ledger period
	Markups Retail	Number(20,4)	markup retail	The retail markups summed from the daily_data table for this stock ledger period
	Markup Cancellations Retail	Number(20,4)	markup can retail	The retail markup cancellations summed from the daily_data table for this stock ledger period
	Clearance Markdowns Retail	Number(20,4)	clear markdown retail	The retail clearance markdowns summed from the daily_data table for this stock ledger period
	Permanent Markdowns Retail	Number(20,4)	perm markdown retail	The retail permanent markdowns summed from the daily_data table for this stock ledger period

Record Name	Field Name	Field Type	Default Value	Description
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	Keeps track of the record's position in the file by line number
	Control Number File Line Count	Number(10)	total detail lines	Sum of all transaction lines, not including file header and trailer

Error File

Standard Retek batch error handling will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

Level Customization:

CLASS LEVEL:

- Remove the subclass field from the export file specification.
- Remove subclass from the logical unit of work, the start string array, and the driving cursor.
- Remove space created for subclass from out_buffer.
- Remove logic involving subclass from the fprintf statement printing to the output file.

MONTH LEVEL (454 & JULIAN):

- Change all variable names like "eow_date" to "eom_date".
- Call CAL_TO_454_LDOM (for 454 calendar) or CAL_TO_CAL_LDOM (for Julian calendar) instead of CAL_TO_454_LDOW to determine the last day of the current month (L_eom_date).
- Call CAL_TO_454_FDOM (for 454 calendar) or CAL_TO_CAL_FDOM (for Julian calendar) to determine the first day of the current month (L_bom_date).
- Change the cursor line "and d.data_date >= L_eow_date - 6" to "and w.eow_date >= L_bom_date".

- Remove the week_no field from the export file specification.
- Remove space created for week_no from out_buffer.
- Remove logic involving week_no from the fprintf statement printing to the output file.
- Change the cursor to select opening stock figures from month_data and summed rows from week_data .

Open to Buy Download [otbdnld]

Design Overview

This batch program will extract current and future Open to Buy data from the OTB table and export it to a flat file for use by an outside planning system.

All Open to Buy data except budget figures is exported directly to the file from the table with no intermediate manipulation.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
OTB	Yes	Yes	No	No	No
PERIOD	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE 4 (weekly/monthly)

Scheduling Diagram: This program must be run after salweek for the week just ended. This program and otbdlsal can run anytime after salweek, but saldly cannot run between otbdnld and otbdlsal

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Since array fetches are used in this program, only one thread will be used.

Restart Recovery

The logical unit of work for the otbdnld module is department, class, subclass and end-of-week date with a recommended commit counter setting of 10,000. Each time the record counter equals the maximum recommended commit number, an application image array record will be written to the restart_start_array for restart/recovery if a fatal error occurs.

```

SELECT      dept,

class,

subclass,

TO_CHAR(eow_date,'YYYYMMDD'),

week_no,

month_no,

half_no,

NVL(cancel_amt,0),

NVL(n_approved_amt,0),

NVL(n_receipts_amt,0),

NVL(b_approved_amt,0),

NVL(b_receipts_amt,0),

NVL(a_approved_amt,0),

NVL(a_receipts_amt,0)

FROM otb

WHERE      eow_date >=

TO_DATE(file_create_date,'YYYYMMDD')

AND  (dept > NVL(:ora_restart_dept, -999) or

(dept = :ora_restart_dept and

      (class > :ora_restart_class or

      (class = :ora_restart_class and

      (subclass > :ora_restart_subclass or

```

```
(subclass = :ora_restart_subclass and
eow_date > :ora_restart_eow_date)
)
)
)
)
)
ORDER BY    dept,
            class,
            subclass,
            eow_date;
```

Program Flow

declare the start string array to include department, class, subclass and end-of-week date

declare output file variables

main()

{

 accept the output file name as an argument from the command line

 init()

 {

 initialize restart/recovery

 copy start string variables from the restart_start_array

 fetch today's date and, if not restarting, select
otb_file_rec_sequence.NEXTVAL

 open the temp output file

 if not restarting, write a header record to the temp file

```
}

declare the driving cursor
table_process()
{
    set up out_buffer to receive fetched data
    open the driving cursor
    LOOP
        fetch
        if no rows are found, set the end flag
        break if there are no rows left to process
    process_records()
        LOOP through the records to be processed
        increment the file line sequence number
        print each record fetched to the temp file
    end LOOP
    break if the end flag has been set
    set the application image array and call restart_commit()
to save it
        call restart_file_write() to write successfully processed
records from the temp file to the final output file
    end LOOP
    close the driving cursor
    write a trailer record to the temp file
}

final()
```

```
{  
    close restart/recovery logic  
    if no errors were encountered in the last round of processing,  
    write any remaining records in the temp file to the final output file  
    close the export file  
    remove the temp file  
}
```

Shared Modules

N/A

Function Level Description

Restart/recovery for table-based processing should be implemented for this program.

main():

- Accept three runtime parameters at the command line: program name, user name/password, and output file name.

init():

- Initialize restart/recovery.
- Open the temp file for output.
- Fetch today's date into file_write_date.
- If not restarting, fetch otb_file_rec_sequence.NEXTVAL into transaction_set_control_number and write a file header record to the temp file.
- Set up an array to receive fetches in the table_process() function - memory should be dynamically allocated so that the cursor fetches arrays of size g_l_restart_max_counter.

table_process():

- Set up out_buffer to receive fetched data:

```
strcpy(out_buffer, "%5s"); /* file type record descriptor */
```

```
strcat(out_buffer, "%-10d"); /* file line sequence number */
```

```

strcat(out_buffer, "%-14d"); /* transaction set control number */

strcat(out_buffer, "%04d"); /* dept */

strcat(out_buffer, "%04d"); /* class */

....

strcat(out_buffer, "%020.0f"); /* cancel_amt */

....

```

Each strcpy/strcat combination should set up space in out_buffer for a single fetched variable. It should copy the appropriate data type (d for decimal integers or s for strings), and all numbers should be left-padded with zeroes except counters, which are left-justified and right-padded with spaces. The length of the space should correspond with the variable's length on the OTB table (e.g. cancel_amt is an oracle variable of (20,4)--it needs a space 20 wide and should be left-padded with zeroes (%020.0f)). The name of the variable that will be copied into that space is included as a comment to ensure maintainability of this code.

- Open the driving cursor C_OTB.
- Use a while(1) loop to fetch rows from the table into the arrays.
- If all rows on the table have been read, set a flag to indicate that processing should end after this array.
- Set file_type_record_descriptor to 'FDETL'.
- Break if there are no more rows to process.
- Call process_records() to process the array.
- Break if the end flag has been set.
- Copy the last row's start string variables into ora_restart_cur_string, delimited by semicolons. This will serve as the application image array. Call restart_commit() to save it.
- Call restart_file_write() to write successfully processed records to the temp file.
- End while(1) loop.
- Close C_OTB.
- Write a trailer record to the temp file with control_number_file_line_count = file_line_sequence_number - 2 and file_type_record_descriptor = 'FTAIL'.

final():

- Close restart/recovery logic.
- If no errors were encountered in the final round of processing, write any remaining records from the temp file to the final output file.
- Close the output file and delete the temp file.

process_records():

- Loop through a subset of records and write each of them to the temp file using `out_buffer` as a format mask.
- The `file_line_sequence_number` should be incremented each time a record is written.

I/O Specification

Output File:

The output file name should be accepted as a runtime parameter at the command line.

The file will be formatted as fixed-width values with no delimiters or punctuation. A new-line character will indicate the end of each record.

A sample detail record would look like the following:

[illegible]

A sequence will need to be added to the database to increment the Transaction Set Control Number. This sequence should be large enough that there is very little danger of running out of numbers. It will give a unique identifying number to every file exported from Retek to an outside planning system.

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	Keeps track of the record's position in the file by line number

Record Name	Field Name	Field Type	Default Value	Description
	File Type Definition	Char(4)	OTBE	Identifies file as 'OTB Export'
	File Create Date	Date	today's date	Date file was written by batch program
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	Keeps track of the record's position in the file by line number
	Transaction Set Control Number	Number(14)	sequence number	Used to force unique file check
	Department	Number(4)	dept identifier	The id number of a department
	Class	Number(4)	class identifier	The id number of a class within the department given
	Subclass	Number(4)	subclass identifier	The id number of a subclass within the class given
	Eow Date	Date	end of week date	The end of week date for the budgeted period
	Week No.	Number(2)	week no.	The week number in the month for the budgeted period
	Month No.	Number(2)	month no.	The month number in the half for the budgeted period
	Half No.	Number(5)	half no.	The half number for the budgeted period
	Cancel Amt.	Number(20,4)	cancel amt.	The total amt. cancelled from orders of all order type for the budgeted period
	N Approved Amt	Number(20,4)	non-basic approved amt	The amt. of approved non-basic orders for the budgeted period

Record Name	Field Name	Field Type	Default Value	Description
	N Receipts Amt	Number(20,4)	non-basic receipts amt	The amt. of non-basic orders due in the budgeted period that have been received
	B Approved Amt	Number(20,4)	buyer-basic approved amt	The amt. of approved buyer-basic orders for the budgeted period
	B Receipts Amt	Number(20,4)	buyer-basic receipts amt	The amt. of buyer-basic orders due in the budgeted period that have been received
	A Approved Amt	Number(20,4)	auto-basic approved amt	The amt. of approved auto-basic orders for the budgeted period
	A Receipts Amt	Number(20,4)	auto-basic receipts amt	The amt. of auto-basic orders due in the budgeted period that have been received
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence Number	Number(10)	line number in file	Keeps track of the record's position in the file by line number
	Control Number File Line Count	Number(10)	total detail lines	sum of all transaction lines, not including file header and trailer

Error File:

Standard Retek batch error handling will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

Level Customization:

CLASS LEVEL (extrapolate for department level):

- Remove the subclass field from the export file specification.
- Remove subclass from the logical unit of work, the start string array, and the driving cursor.
- Remove space created for subclass from out_buffer.
- Remove logic involving subclass from the fprintf statement printing to the output file.

MONTH LEVEL (454 & JULIAN):

- Change all variable names like 'eow_date' to 'eom_date'.
- Remove the week_no field from export file specification.
- Remove space created for week_no from out_buffer.
- Remove logic involving week_no from the fprintf statement printing to the output file.

Open To Buy Purge [otbprg]

Design Overview

This batch program runs at the end of the half to delete rows from the OTB table that are at least one half old. The current and previous half's purchase budget data is retained. Open to Buy history can be retained longer with a modification to the function that calculates the purge date.

Table Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
OTB	No	No	No	No	Yes

Scheduling Constraints

Processing Cycle: PHASE AD-HOC (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

There is no restart/recovery in this module.

The logical unit of work is 10,000 records of data less than a particular date.

There is no driving cursor in this module.

Program Flow

Aside from the main function and its standard sub-functions, the body of this program is all in one function called process() which does the following:

- Retrieves the purge date.
- Deletes records with an end-of-week date prior to the purge date.
- Commits every 10,000 records.

Shared Modules

OTB_SQL.GET_PURGE_DATE is called to determine which records should be deleted. Deleted records should have a week ending date that is earlier than the purge date.

SQL_LIB.BATCH_MSG is needed to format error messaging from the package so that the batch module can use the information.

Function Level Description

process():

Call the function OTB_SQL.GET_PURGE_DATE to retrieve the date that will be used to determine which records should be purged.

Set up a while loop to delete from the OTB table in groups of 10,000 records where the end-of-week date is less than or equal to the OTB purge date. After each set of 10,000 records is deleted, perform a commit. If less than 10,000 records were processed by the previous delete statement, exit the loop.

Error handling should be written according to the new standards for batch programs calling a batch-enabled package.

I/O Specification

N/A

Technical Issues

N/A

Accept Forward Limit Percentages Upload [otbupfwd]

Design Overview

The purpose of this batch module is to accept forward limit percentages by period from an outside planning system. This data is then inserted into or used to update the forward limit percentage held on the OTB_FWD_LIMITS table.

There is no processing done to these records as the data is transferred directly from the input file to the table. If there is not a row on the table to update, a new row is inserted with the department, class, subclass, period ahead and forward limit percentage as taken from the input file.

Tables affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
OTB_FWD_LIMIT	No	No	Yes	Yes	No

Scheduling Constraints

Processing Cycle: PHASE AD-HOC

Scheduling Diagram: This program is independent of all other batch processing and can be run at any time in the batch cycle.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

The logical unit of work for the otbupfwd module is the discrete forward limit transaction. Each record is uniquely identified by department, class, subclass and period ahead. Processing of each row is independent and thus if an erroneous record is found during processing, only that record needs to be corrected and reprocessed.

Error handling will also be based on this logical unit of work. If a record fails validation, it will be written to a rejected record file. This file will facilitate easy reprocessing once the error is fixed by writing the record exactly as it was in the source file. All errors will be written to an error log file using the `WRITE_ERROR` library function.

The recommended commit counter setting is 10,000. A savepoint will be issued after each record is successfully processed. If a non-fatal error occurs, a rollback to the last savepoint will be issued so that the rejected record is not posted to the database. If a fatal error occurs and restart is necessary, processing will restart at the last commit point.

Program Flow

```
main()
{
    accept the names of the input and reject files as arguments from the
    command line

    declare structures for the file header, detail and trailer records

    declare an application image array that includes g_l_rec_cnt and
    g_l_rej_cnt

    init()
    {
        open the input and reject files

        initialize restart/recovery

        call get_record()

        if it fails, return FATAL ERROR
    }
}
```

```
file_process()
{
    LOOP
        read the number of records indicated by g_l_restart_max_ctr into
memory
        if the end of file character is reached, decrement the record
counter by 1 and set the end of file flag to TRUE
        LOOP
            read a record
            copy items from structs into varchar arrays and
nullpad them
            validate_record()
            check that department, class, subclass
and period ahead are not null by calling field_has_value()
            check that department, class and
subclass are numbers by calling valid_all_numeric() and call
SUBCLASS_VALIDATE_SQL.EXIST to see that they exist on the subclass
table
            check that 0.00 <= fwd_limit_pct <= 100.00
            check that 0 < period_ahead < 100
            if the record fails validation, return
NON-FATAL ERROR (the library validation functions will return 1 on NON-
FATAL ERROR)
            if NON-FATAL ERROR is returned
during validation
            call write_to_rej_file() to write the
invalid record to the reject file
            increment g_l_rej_cnt
            move on to the next record
        process_record()
```

```

                                update the appropriate row on
OTB_FWD_LIMITS with fwd_limit_pct from the input file record

                                if no row is found to update, insert a
new one with the input file record values

                                if the update/insert fails, return FATAL
ERROR

                                end LOOP

                                increment the record counter by the number of records
processed in the loop

                                set the application image array

                                call restart_file_commit()

                                call restart_file_write()

                                exit if the end of file flag has been set to TRUE

                                end LOOP

                                call process_dtl_ftail () to process the file trailer record
}

final()
{
    close restart/recovery logic

    close the reject file
}
}

```

Shared Modules

SUBCLASS_VALIDATE_SQL.EXIST: This function will validate that a department/class/subclass combination exists on the subclass table.

Function Level Description

main():

Accept four runtime parameters at the command line: program name, user name/password, input file name, and reject file name. Copy the input reject file names into strings for later use.

init():

Open the input and reject files.

Initialize restart/recovery.

Call the get_record() function to read the file header record.

file_process():

Set up a buffer in memory to hold the number of input file rows indicated by g_l_restart_max_counter.

Use a while loop to read g_l_restart_max_counter number of rows into the buffer.

If the end of file character is reached, decrement the record counter by one (to avoid processing file trailer record) and set the end-of-file flag to TRUE.

Set up a FOR loop to read each record from the buffer into the detail record structure and process it.

Copy each record item into a local variable for processing.

Call the validate_record() function to validate record items.

If validate_record() returns a value of 1 (an error was encountered) then handle NON-FATAL ERROR.

Call the process_record() function.

If record processing fails, handle FATAL ERROR.

End FOR loop.

Increment the record counter by the number of records processed in the loop.

Set the application image array for restart/recovery and call restart_file_commit() to save it.

Call the restart_file_write() function to write rejected records to the rejected record file.

If the end-of-file has been reached, exit the while loop.

End while loop.

Call the process_dtl_ftail() function to process the file trailer record.

validate_record():

Check that the period ahead is a number between 1 and 99 (inclusive)

Ensure that the department, class, subclass and period ahead fields all contain values by calling field_has_value() for each. If one of the fields is NULL, field_has_value() will return 1 and the record should be written to the rejected record file.

Ensure that the department, class and subclass fields are numeric by calling valid_all_numeric() for each and, if no error is returned, call SUBCLASS_VALIDATE_SQL.EXIST to validate that the record's department/class/subclass combination exists on the subclass table.

Check that forward limit percent is a number between 0.00 and 100.00 (inclusive).

process_record():

Update the OTB_FWD_LIMITS table (by department, class, subclass and period ahead) with the forward limit percent extracted from the input file.

If no row is found to update, insert a new one with the input record's values.

If the insert fails, raise FATAL ERROR.

final():

Close restart/recovery logic.

Close the reject file.

on FATAL ERROR:

Use standard batch error handling to write to the error log file.

Rollback to the previous physical commit point.

Exit the program.

on NON-FATAL ERROR:

Rollback to the previous savepoint.

Call the function write_to_rej_file() to write rejected records to the rejected record file.

Increment g_l_rej_cnt.

I/O Specification

Input File:

The input file name should be accepted as a runtime parameter at the command line.

The file will be formatted as fixed-width values with no delimiters or punctuation. A new-line character will indicate the end of each record.

A sample detail record would look like the following:

FDETL12345678901234567890123412341234123499123456789012

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence Number	Number(10)	Line number in file	Keeps track of the record's position in the file by line number
	File Type Definition	Char(4)	FWDI	Identifies file as 'FWD Import'
	File Create Date	Date	create date	Date file was written by external system
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Sequence Number	Number(10)	Line number in file	Keeps track of the record's position in the file by line number
	Transaction Set Control Number	Number(14)	specified by external system	Used to force unique transaction check
	Department	Number(4)	dept identifier	The id number of a department
	Class	Number(4)	class identifier	The id number of a class within the department given
	Subclass	Number(4)	subclass identifier	The id number of a subclass within the class given

Record Name	Field Name	Field Type	Default Value	Description
	Period Ahead	Number(2)	period identifier	The period ahead (1-99) for the forward limit planning
	Forward Limit Percent	Number(12)	planning percentage	Percentage of period's budget available to buyer at current time
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence Number	Number(10)	Line number in file	Keeps track of the record's position in the file by line number
	Control Number File Line Count	Number(10)	total detail lines	sum of all transaction lines, not including file header and trailer

Reject File

The reject file will be formatted identically to the input file for easy re-processing. Library functions will create the header and trailer records as well as keeping track of the rejected record count. A reject file will be created in all cases. If no errors occur, the reject file will consist only of file header and trailer records, and the file line count will be equal to 0. The reject file name should be specified as a runtime parameter.

Error File

Standard Retek batch error handling will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

Level Customization:

CLASS LEVEL (extrapolate for department level):

Remove the subclass field from the input file specification.

Remove subclass from the file detail structure.

Remove local variable designated to hold subclass from the declarations above file_process().

In file_process(), remove logic copying subclass from memory to a local variable.

In validate_record(), remove logic validating that subclass is numeric and not blank.

In validate_record(), call CLASS_VALIDATE_SQL.EXISTS instead of SUBCLASS_VALIDATE_SQL.EXISTS.

In process_record(), remove subclass from all update or insert logic.

454 MONTH LEVEL

No modifications necessary.

JULIAN MONTH LEVEL

No modifications necessary.

New Budget Data And Budget Adjustments [otbupld]

Design Overview

The purpose of this batch module is to accept new budget data and budget adjustments from an outside planning system. This data is then inserted into or used to update the budget figures on the OTB table.

Detail processing is very limited. If there is not a row on the table to update, a new row is inserted with the department, class, subclass, end-of-week date and budget amount for the specified order type as taken from the input file. Order types tracked by Retek are Non-Basic (N/B), Buyer Reorder of Basic (BRB) and Automatic Reorder of Basic (ARB). The date procedures CAL_TO_454 and CAL_TO_454_HALF are used to calculate the half, month and week numbers of the record's end-of-week date, which are also held on the OTB table. If a budget amount is being adjusted, the old budget amount for the specified department, class, subclass, end-of-week date and order type is overwritten with the budget amount extracted from the input file.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
OTB	No	No	Yes	No	No

Scheduling Constraints

Processing Cycle: PHASE AD-HOC

Scheduling Diagram: This program is independent of all other batch processing and can be run at any time in the batch cycle.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

The logical unit of work for the otbupld module is the discrete budget transaction. Each row is uniquely identified by order type, department, class, subclass and end-of-week date. Processing of each row is independent and thus if an erroneous record is found during processing, only that record needs to be corrected and reprocessed.

Error handling will also be based on this logical unit of work. If a record fails validation, it will be written to a rejected record file. This file will facilitate easy reprocessing once the error is fixed by writing the record exactly as it was in the source file. All errors will be written to an error log file using the `WRITE_ERROR` library function.

The recommended commit counter setting is 10,000. A savepoint will be issued after each record is successfully processed. If a non-fatal error occurs, a rollback to the last savepoint will be issued so that the rejected record is not posted to the database. If a fatal error occurs and restart is necessary, processing will restart at the last commit point.

Program Flow

```
main()
{
    accept the names of the input and reject files as arguments from the
    command line

    declare structures for the file header, detail and trailer records

    declare an application image array that includes g_l_rec_cnt and
    g_l_rej_cnt

    init()
    {
        open the input and reject files

        initialize restart/recovery

        call the library function get_record()

        if this fails, return FATAL ERROR
    }
}
```

```

file_process()
{
    LOOP
        read number of records indicated by g_l_restart_max_ctr
into memory

    if the end of file character is reached then decrement the counter by 1 and set the
end of file flag to TRUE

    LOOP
        read a record

        copy items from structs into varchar arrays and nullpad them

        validate_record()

            check that order type, department, class,
subclass and end-of-week date are not NULL by calling field_has_value()

            check that order type is 'A', 'B', or 'N'

            check that the end-of-week date is a valid date by calling valid_date()
and a valid end-of-week date by calling CAL_TO_454_LDOW

            check that department, class and
subclass are numbers by calling valid_all_numeric() and call
SUBCLASS_VALIDATE_SQL.EXIST to check that they exist on the subclass
table

            check that the budget amount is a number by
calling valid_all_numeric_signed()

            if record fails validation, return NON-FATAL
ERROR

            if NON-FATAL ERROR is returned
during validation

            call write_to_rej_file() to write the
invalid record to the reject file

            increment g_l_rej_cnt

            move on to the next record

        process_record()

```

```
        call OTB_SQL.OTB_INSERTS
    if the function returns FALSE then return FATAL ERROR
    end LOOP
    increment the record counter by the number of records processed in the
loop
        set the application image array
        call the restart file commit process
        call restart_file_write()
        if the end of file flag = TRUE then exit
    end LOOP
    call process_dtl_ftail () to process the file trailer record
}

final()
{
    close restart/recovery logic
    close the reject file
}
}
```


Shared Modules

SUBCLASS_VALIDATE_SQL.EXIST: This function will validate that a department/class/subclass combination exists on the subclass table.

CAL_TO_454_LDOW: This procedure will return an end-of-week date that can be used to validate the input record's end-of-week date.

OTB_SQL.OTB_INSERTS: This function performs the inserts into and updates of the OTB table. If there is no record on the OTB table for the specified department, class, subclass and end-of-week date, this package calculates the half, month and week numbers for the given end-of-week date using CAL_TO_454_HALF and CAL_TO_454. It then inserts a new row with the data from the input file and the calculated date figures with all other rows assigned a value of zero. If there is a row already on the table, the specified order type's budget amount is overwritten with the number extracted from the input file.

Function Level Description

main():

Accept four runtime parameters at the command line: program name, user name/password, input file name, and reject file name. Copy the input and reject file names into strings for later use.

init():

Open the input and reject files.

Initialize restart/recovery.

Call the get_record() function to read the input file header record

file_process():

Set up a buffer in memory to hold the number of input file rows indicated by g_l_restart_max_counter.

Use a while loop to read g_l_restart_max_counter number of rows into the buffer.

If the end of file character is reached, decrement the record counter by one (to avoid processing file trailer record) and set the end-of-file flag to TRUE.

Set up a FOR loop to read each record from the buffer into the detail record structure and process it.

Copy each record field into a local variable for processing.

Call the `validate_record()` function to validate record fields.

If `validate_record()` returns a value of 1 (an error was encountered) then handle NON-FATAL ERROR.

Call the `process_record()` function

If record processing fails, handle FATAL ERROR.

End FOR loop.

Increment the record counter by the number of records processed in the loop.

Set the application image array for restart/recovery and call the `restart_file_commit()` function to save it.

Call the `restart_file_write()` function to write the rejected records to the rejected record file.

If the end-of-file flag has been set to TRUE, exit the while loop.

End while loop.

Call the `process_dtl_ftail()` function to process the file trailer record

`validate_record()`:

Ensure that the order type, department, class, subclass, and end-of-week date fields all contain values by calling the function `field_has_value()` for each. If one of the fields is NULL, `field_has_value()` will return 1 and the record should be written to the reject file.

Check that the order type field contains 'A', 'B' or 'N'.

Check that the input record's end-of-week date is a valid date in the correct format by calling the `valid_date()` function. If not, `valid_date()` will return 1 and the record should be written to the reject file. If `valid_date()` returns 0, check that the end-of-week date is a valid end-of-week date using `CAL_TO_454_LDOW`.

Ensure that the department, class and subclass are all numeric by calling `valid_all_numeric()` for each and, if no error is returned from that function, call `SUBCLASS_VALIDATE_SQL.EXIST` to validate that the record's department/class/subclass combination exists on the subclass table.

Check that the budget amount is numeric by calling `valid_all_numeric_signed()`.

`process_record()`:

Call `OTB_SQL.OTB_INSERTS` to insert or update the OTB table.

`final()`:

Close restart/recovery logic.

Close the reject file.

on FATAL ERROR:

Use standard batch error handling to write to the error log file.

Rollback to previous physical commit point.

Exit the program.

on NON-FATAL ERROR:

Rollback to the previous savepoint.

Call the function `write_to_rej_file()` to write the rejected record to the rejected record file.

Increment `g_l_rej_cnt`.

I/O Specification

Input File:

The input file name should be accepted as a runtime parameter at the command line.

The file will be formatted as fixed-width values with no delimiters or punctuation. A new-line character will indicate the end of each record.

The end-of-week date must be passed in the format 'YYYYMMDD' in order to validate it using `valid_date()`.

A sample detail record would look like the following:

FDETL123456789012345678901234A123412341234199501151234567890123456789

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence Number	Number(10)	Line number in file	Keeps track of the record's position in the file by line number

Record Name	Field Name	Field Type	Default Value	Description
	File Type Definition	Char(4)	OTBI	Identifies file as 'OTB Import'
	File Create Date	Date	create date	Date file was written by external system
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Sequence Number	Number(10)	Line number in file	Keeps track of the record's position in the file by line number
	Transaction Set Control Number	Number(14)	specified by external system	Used to force unique transaction check
	Order Type	Char(1)	A, B, N	Order type budgeted for: specified as A for auto-basic replenishment, B for buyer-basic replenishment, and N for non-basic orders
	Department	Number(4)	dept identifier	The id number of a department
	Class	Number(4)	class identifier	The id number of a class within the department given
	Subclass	Number(4)	subclass identifier	The id number of a subclass within the class given
	Eow Date	Date	end of week date	The end of week date for the budgeted week (formatted as 'YYYYMMDD')
	Budget Amount	Number(20)	budget amount	Budgeted amount for the specified order type/week
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type

Record Name	Field Name	Field Type	Default Value	Description
	File Line Sequence Number	Number(10)	Line number in file	Keeps track of the record's position in the file by line number
	Control Number File Line Count	Number(10)	total detail lines	sum of all transaction lines, not including file header and trailer

Reject File:

The reject file will be formatted identically to the input file for easy re-processing. Library functions will create the header and trailer records as well as keeping track of the rejected record count. A reject file will be created in all cases. If no errors occur, the reject file will consist only of file header and trailer records, and the file line count will be equal to 0. The reject file name should be specified as a runtime parameter.

Error File:

Standard Retek batch error handling will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

Level Customization:

CLASS LEVEL (extrapolate for department level):

Remove the subclass field from the input file specification.

Remove subclass from the file detail structure.

Remove the local variable designated to hold subclass from the declarations above file_process().

In file_process(), remove the logic copying subclass from memory to a local variable.

In validate_record(), remove all logic validating that subclass is numeric.

In validate_record(), call CLASS_VALIDATE_SQL.EXISTS instead of SUBCLASS_VALIDATE_SQL.EXISTS.

In process_record(), remove subclass from the call to OTB_SQL.OTB_INSERTS.

454 MONTH LEVEL:

Change all variable names like 'eow_date' to eom_date.

Call CAL_TO_454_LDOME to check that the input record's end-of-month date is a valid 454 calendar end-of-month date.

JULIAN MONTH LEVEL:

Change all variable names like 'eow_date' to eom_date.

Call CAL_TO_CAL_LDOME to check that the input record's end-of-month date is a valid Julian calendar end-of-month date.

Clearance Pricing POS Download [pccdnld]

Design Overview

The purpose of the Clearance Download (pccdnld) module is to send clearance markdown retail prices to the point of sales system. Clearance markdowns that are to take effect within the predetermined number of days are written out to the POS_MODS table, which will be used as an interface point with the point of sale. Also, the clearance detail records are updated with the current date as a downloaded date. The number of days prior to taking effect that clearance markdowns are downloaded is maintained as a system option.

Table	Index	Select	Insert	Update	Delete
PERIOD	No	Yes	No	No	No
UNIT_OPTIONS	No	Yes	No	No	No
CLEAR_SUSP_DETAIL	No	Yes	No	Yes	No
CLEAR_SUSP_HEAD	No	Yes	No	No	No
WIN_SKUS	Yes	Yes	No	No	No
WIN_STORE	Yes	Yes	No	No	No
RAG_STYLE	Yes	Yes	No	No	No
RAG_STYLE_ST	Yes	Yes	No	No	No
RAG_SKUS_ST	Yes	Yes	No	No	No
POS_MODS	No	No	Yes	No	No
UPC_EAN	Yes	Yes	No	No	No

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: Phase 1

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Clearance

Restart Recovery

Driving Cursor

```
EXEC SQL DECLARE c_clear CURSOR FOR

SELECT csd.sku,

       csd.clearance,

       TO_CHAR(csd.active_date, 'DDMMYYYY'),

       csd.zone_group_id,

       csd.unit_retail,

       pzgs.store,

       csd.rowid,

       ';' || TO_CHAR(csd.clearance) ||

       ';' || TO_CHAR(csd.sku) ||

       ';' || TO_CHAR(csd.zone_group_id) ||

       ';' || TO_CHAR(csd.zone_id)

FROM clear_susp_detail csd,

     clear_susp_head csh,

     price_zone_group_store pzgs,

     v_restart_clearance rv

WHERE :prc_ext_days >= (csd.active_date - TO_DATE(:vdate,
'DDMMYYYY'))

AND csd.active_date >= TO_DATE(:vdate, 'DDMMYYYY')

AND csh.status      = 'A'

AND csd.downloaded_date is NULL

AND csd.clearance   = csh.clearance

AND pzgs.zone_group_id = csd.zone_group_id

AND pzgs.zone_id     = csd.zone_id
```



```
AND rv.driver_value = csh.clearance
AND rv.driver_name = :ora_restart_driver_name
AND rv.num_threads = :ora_restart_num_threads
AND rv.thread_val = :ora_restart_thread_val
ORDER BY csd.clearance,
        csd.sku,
        csd.zone_group_id,
        csd.zone_group_id;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Retek Clearance Pricing Update [pccext]

Design Overview

The function of this program is to extract the new clearance event retail pricing from the clearance suspense tables and update the retail information in Retek. Clearance events that are to take place on the following business day are selected and the appropriate sku/location tables are updated with the new pricing information and with a new clearance indicator value. Transaction-level stock ledger, price history, supplier history and clearance sell through records are also written. If a SKU/store is on replenishment, using the RMS replenishment functionality, then when the SKU/store goes on clearance, it is taken off of replenishment. If all locations for a SKU are no longer on replenishment, then the replenishment indicator at the item level is also updated in this program. This program is run nightly.

Table	Index	Select	Insert	Update	Delete
PERIOD	No	Yes	No	No	No
CLEAR_SUSP_DETAIL	No	Yes	No	No	No
CLEAR_SUSP_HEAD	No	Yes	No	Yes	No
WIN_SKUS	Yes	Yes	No	Yes	No
WIN_WH	Yes	No	No	Yes	No
WIN_STORE	Yes	Yes	No	Yes	No
RAG_STYLE	Yes	Yes	No	Yes	No
RAG_SKUS	Yes	Yes	No	No	No
RAG_SKUS_WH	Yes	No	No	Yes	No
RAG_SKUS_ST	Yes	Yes	No	Yes	No
CLEAR_SELL_THRGH	No	No	Yes	No	No
TRAN_DATA	No	No	Yes	No	No
SUP_DATA	No	No	Yes	No	No
PRICE_HIST	No	No	Yes	No	No
PRICE_ZONE_GROUP_STORE	Yes	Yes	No	No	No
CLEAR_RESET_CALC	No	Yes	No	No	No
SKU_SUPPLIER	Yes	Yes	No	No	No

Indexes: WIN_SKUS(sku),
 WIN_WH(sku),
 WIN_STORE(sku), (sku, store),
 RAG_STYLE(style),
 RAG_SKUS(style),
 RAG_SKUS_WH(style),
 RAG_SKUS_ST(style, store),
 PRICE_ZONE_GROUP_STORE(zone_group_id, zone_id),
 (zone_group_id, store),
 SKU_SUPPLIER(sku)

Scheduling Constraints

Processing Cycle: Phase 3
 Scheduling Diagram: N/A
 Pre-Processing: pctrans/pctranf should complete before the pccext module begins.
 Post-Processing: pccext should not start until pccext has completed.
 Threading Scheme: Clearance

Restart Recovery

```
EXEC SQL DECLARE c_clear CURSOR FOR
SELECT csh.clearance,
       csh.reason,
       csh.status,
       csd.sku,
       csd.unit_retail,
       TO_CHAR(csd.active_date, 'DDMMYYYY'),
       pzgs.store,
```

```
        '||TO_CHAR(csh.clearance)||
        '||TO_CHAR(csd.sku)||
        '||TO_CHAR(pzgs.store)
FROM v_restart_clearance rv,
     price_zone_group_store pzgs,
     clear_susp_detail csd,
     clear_susp_head csh
WHERE csh.status      = 'A'
      AND csh.clearance = csd.clearance
      AND csd.active_date = to_date(:ps_tomorrow, 'DDMMYYYY')
      AND csd.zone_group_id = pzgs.zone_group_id
      AND csd.zone_id      = pzgs.zone_id
      AND rv.driver_value  = csh.clearance
      AND rv.driver_name   = :ora_restart_driver_name
      AND rv.num_threads   = :ora_restart_num_threads
      AND rv.thread_val    = :ora_restart_thread_val
      AND (csh.clearance > NVL(:ora_restart_clearance, -999) OR
           (csh.clearance = :ora_restart_clearance AND
            (csd.sku > :ora_restart_sku OR
             (csd.sku = :ora_restart_sku AND
              (pzgs.store > :ora_restart_store))))))
ORDER BY csh.clearance, sku, store;
```

Program Flow

N/A

Shared Modules

GET_SYSTEM_IND

STKLEDGR_SQL.TRAN_DATA_INSERT

PRCCHGTYP

GET_TRANSFER_QTY

REPLENISHMENT_SQL.ON_REPLENISHMENT - called to determine if the item (i.e. staple SKU, fashion style) is still on replenishment after its locations have been taken off of replenishment when the locations go on clearance. If no locations are on replenishment, the replenishment indicator at the item level is set equal to No.

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Clearance Event Deletions [pccprg]

Design Overview

The Clearance Purge (pccprg) module removes completed, canceled, and rejected clearance events from the system. Before a clearance event can be deleted, a pre-determined number of months must have passed since the reset date of the clearance. The pre-determined number of months is based on the value entered for the system option Clearance Retention Months.

Tables affected:

Tables	Index	Select	Insert	Update	Delete
CLEAR_RESET_CALC	No	Yes	No	No	Yes
CLEAR_SELL_THROUGH	No	No	No	No	Yes
CLEAR_SUSP_DTL	No	Yes	No	No	Yes
CLEAR_SUSP_HEAD	No	Yes	No	No	Yes
PERIOD	No	Yes	No	No	No
PRICE_EVENT_ZONE	No	No	No	No	Yes
PRICE_EVENT_SKU	No	No	No	No	Yes
UNIT_OPTIONS	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Ad Hoc

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Clearance Reset Pricing POS Extract [pccrdnld]

Design Overview

The Clearance Reset Pricing Information (pccrdnld) module is used to send clearance reset pricing information to the point of sale system. When a clearance event is within a pre-determined number of days of its reset date, this program will gather current clearance pricing information and the pricing information to which an item will be reset. This information will be written out to a table that will be used as an interface point with the point of sale. Clearance records will be updated to indicate that they have had reset prices downloaded to the point of sale by setting a reset downloaded date with the current date.

Tables	Index	Select	Insert	Update	Delete
PERIOD	No	Yes	No	No	No
UNIT_OPTIONS	No	Yes	No	No	No
CLEAR_RESET_CALC	No	Yes	No	No	No
CLEAR_SUSP_HEAD	No	Yes	No	No	No
WIN_SKUS	Yes	Yes	No	Yes	No
WIN_STORE	Yes	Yes	No	Yes	No
RAG_STYLE	Yes	Yes	No	Yes	No
RAG_STYLE_ST	Yes	Yes	No	Yes	No
RAG_SKUS_ST	Yes	Yes	No	No	No
POS_MODS	No	No	Yes	No	No
UPC_EAN	Yes	Yes	No	No	No
ITEM_ZONE_PRICE	Yes	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Phase 1

Scheduling Diagram: N/A

Pre-Processing: pcdnld should complete processing before this module begins

Post-Processing: N/A

Threading Scheme: Clearance

Restart Recovery

The driving cursor was changed to get the system_indicator directly using a join to the desc_look table instead of calling the GET_SYS_IND function:

```
EXEC SQL DECLARE c_get_reset CURSOR FOR

SELECT crc.clearance,

       crc.sku,

       TO_CHAR(crc.reset_date, 'DDMMYYYY'),

       pzgs.store,

       ROWIDTOCHAR(crc.rowid),

       dl.system_ind,

       ';' || TO_CHAR(crc.clearance) ||

       ';' || TO_CHAR(crc.sku) ||

       ';' || TO_CHAR(crc.zone_group_id) ||

       ';' || TO_CHAR(crc.zone_id)

FROM clear_reset_calc crc,

     clear_susp_head csh,

     price_zone_group_store pzgs,

     desc_look dl,

     v_restart_clearance rv

WHERE crc.reset_date is NOT NULL

AND (

     (csh.status = 'C'

      AND EXISTS (SELECT 'x'

                  FROM clear_susp_detail csd

                  WHERE csd.clearance = csh.clearance

                  AND csd.sku = crc.sku

                  AND csd.sku = dl.sku
```

```

        AND csd.zone_group_id = crc.zone_group_id
        AND csd.zone_id = crc.zone_id
        AND csd.downloaded_date is NOT NULL))

    OR

    (csh.status = 'A'

    AND crc.sku = dl.sku

    AND :prc_ext_days >= (crc.reset_date -
to_date(:ps_vdate,'DDMMYYYY'))
    )

    AND csh.clearance = crc.clearance

    AND crc.zone_group_id = pzgs.zone_group_id

    AND crc.zone_id = pzgs.zone_id

    AND crc.reset_downloaded_date is NULL

    AND rv.driver_value      = csh.clearance

    AND rv.driver_name       = :ps_restart_driver_name

    AND rv.num_threads       = :pi_restart_num_threads

    AND rv.thread_val        = :pi_restart_thread_val

    ORDER BY crc.clearance,

           crc.sku,

           crc.zone_group_id,

           crc.zone_id;

```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

The call to the `get_sys_ind` function is removed and the system indicator is fetched back in the driving cursor.

I/O Specification

N/A

Technical Issues

N/A

Retek Clearance Reset Price Update [pccrext]

Design Overview

The function of this program is to reset the SKU/store retail prices back to the regular retail prices. Clearance events with reset dates on the following business day are selected and the appropriate sku/location tables are updated with the regular retail price and a new clearance indicator value. Transaction-level stock ledger, price history, supplier history and clearance sell through records are also written.

Tables	Index	Select	Insert	Update	Delete
PERIOD	No	Yes	No	No	No
CLEAR_SUSP_DETAIL	No	Yes	No	No	No
CLEAR_SUSP_HEAD	No	Yes	No	Yes	No
WIN_SKUS	Yes	Yes	No	No	No
WIN_WH	Yes	No	No	Yes	No
WIN_STORE	Yes	Yes	No	Yes	No
RAG_STYLE	Yes	Yes	No	No	No
RAG_SKUS	Yes	Yes	No	No	No
RAG_SKUS_WH	Yes	No	No	Yes	No
RAG_SKUS_ST	Yes	Yes	No	Yes	No
TRAN_DATA	No	No	Yes	No	No
SUP_DATA	No	No	Yes	No	No
PRICE_HIST	No	No	Yes	No	No
ITEM_ZONE_PRICE	Yes	Yes	No	No	No
PRICE_ZONE_GROUP_STORE	Yes	Yes	No	No	No
CLEAR_RESET_CALC	No	Yes	No	No	No
SKU_SUPPLIER	Yes	Yes	No	No	No

Indexes: WIN_SKUS(sku),
WIN_WH(sku),
WIN_STORE(sku), (sku, store),
RAG_STYLE(style),
RAG_SKUS(style),
RAG_SKUS_WH(style),
RAG_SKUS_ST(style, store),
ITEM_ZONE_PRICE(zone_group_id, zone_id),
PRICE_ZONE_GROUP_STORE(zone_group_id, zone_id),
(zone_group_id, store),
SKU_SUPPLIER(sku)

Scheduling Constraints

Processing Cycle: Phase 3
Scheduling Diagram: N/A
Pre-Processing: pcontext and pcontext must complete before this module begins
Post-Processing: N/A
Threading Scheme: Clearance

Restart Recovery

```
EXEC SQL DECLARE c_reset CURSOR FOR  
SELECT csh.clearance,  
csh.status,  
csh.reason,  
crc.sku,  
crc.zone_group_id,  
crc.zone_id,
```

```
        TO_CHAR(crc.reset_date, 'DDMMYYYY'),
        pzgs.store,
        ';' || TO_CHAR(csh.clearance) ||
        ';' || TO_CHAR(crc.sku) ||
        ';' || TO_CHAR(pzgs.store)
FROM v_restart_clearance rv,
     price_zone_group_store pzgs,
     clear_reset_calc crc,
     clear_susp_head csh
WHERE ((csh.status = 'A'
AND   crc.reset_date = to_date(:tomorrow, 'DDMMYYYY'))
OR   (csh.status = 'C'
AND   EXISTS (SELECT 'x'
              FROM clear_susp_detail csd
              WHERE csd.clearance = csh.clearance
              AND   csd.sku = crc.sku
              AND   csd.zone_group_id = crc.zone_group_id
              AND   csd.zone_id = crc.zone_id
              AND   csd.downloaded_date is not null)))
AND   csh.clearance = crc.clearance
AND   crc.zone_group_id = pzgs.zone_group_id
AND   crc.zone_id = pzgs.zone_id
AND   rv.driver_value = csh.clearance
AND   rv.driver_name = :ora_restart_driver_name
AND   rv.num_threads = :ora_restart_num_threads
AND   rv.thread_val = :ora_restart_thread_val
```

```
AND (csh.clearance > NVL(:ora_restart_clearance, -999)
    OR (csh.clearance = :ora_restart_clearance
    AND (crc.sku > :ora_restart_sku OR (crc.sku = :ora_restart_sku
    AND (pzgs.store > :ora_restart_store))))))
ORDER BY csh.clearance, sku, store;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Price Change POS Extract [pcdnld]

Design Overview

The Price Change Download (pcdnld) module selects permanent price change records, which are set to go into effect a pre-determined number of days after today. The details of these records are then written out to the POS_MODS table, which is used as an interface point to with the point of sale system. The number of days before the price change goes into effect that the details are downloaded to the POS system is determined by the system option pos_extract_days on UNIT_OPTIONS. After this program has processed a price change, the status of the price change is changed to Extracted.

Table	Index	Select	Insert	Update	Delete
PERIOD	No	Yes	No	No	No
UNIT_OPTIONS	No	Yes	No	No	No
PRICE_SUSP_HEAD	Yes	Yes	No	Yes	No
PRICE_SUSP_DETAIL	Yes	Yes	No	Yes	No
PRICE_CRITERIA_HEAD	Yes	Yes	No	Yes	No
DESC_LOOK	Yes	Yes	No	No	No
V_RESTART_PRICE_CHG	Yes	Yes	No	No	No
PRICE_ZONE_GROUP_STORE	Yes	Yes	No	No	No
SKU_ZONE_PRICE	Yes	Yes	No	No	No
POS_MODS	No	No	Yes	No	No
UPC_EAN	Yes	Yes	No	No	No
CLEAR_SUSP_HEAD	Yes	Yes	No	No	No
CLEAR_SUSP_DETAIL	Yes	Yes	No	No	No
CLEAR_RESET_CALC	Yes	Yes	No	No	No

Indexes: UPC_EAN (upc), (ean)

Scheduling Constraints

Processing Cycle:	Phase 1
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	Pccrdnld should run only after this module has completed
Threading Scheme:	Price change

Restart Recovery

The restart string is price change + SKU + store.

Records will be fetched from an outer loop into arrays (sized according to restart_max_ctr on RESTART_CONTROL) and each record will be individually processed in a loop. Commits will occur after each record in the for-loop has been processed and before the next set of records are fetched into the arrays. This can happen because status is part of the select criteria on both the price change header and detail tables.

Program Flow

N/A

Shared Modules

TICKET_SQL.PRICE_CHANGE

Function Level Description

init()

This function will, in addition to performing standard restart tasks, fetch the current date from the period table and the number of days buffer required by the point of sales system for pricing modifications (pos_extract_days).

process()

This function selects the impending price change records (price change + SKU + store combinations) for processing. The records are selected into arrays (size determined by commit_max_ctr on the RESTART_CONTROL table) and then each record in the array is processed by a for-loop.

Inside this internal loop the SKU information is retrieved from the DESC_LOOK table (system_ind, dept, & desc_up). If the item is a SKU then process_sku is called, otherwise style items will call the process_style function. If the price change number has changed since the last check then the update_head_status() function is called to update the header status. When the PRICE_SUSP_DETAIL rowid changes then a row is added to an array that will update the status of the PRICE_SUSP_DETAIL table.

After the internal loop has processed the individual records, the pos_mods_insert() function is called to insert POS_MODS records created in the process_sku() function and the price_susp_detail rows that have completed processing are updated by calling the psd_update() function.

process_sku()

This function will select the current/old retail pricing information from SKU_ZONE_PRICE. First it is determined whether the item is on clearance at the price change location by calling the item_on_clearance() function. If it is, then the function is exited (price information is not retrieved and a POS_MODS record is not written). If the price change is a regular price change then the information can be selected using the zone information (zone_id & zone_group_id) on the price_change tables. Otherwise, the “old” zone information must be retrieved from the PRICE_ZONE_GROUP_STORE table based on the store location of the price change. This table is joined with the SKU_ZONE_PRICE table to retrieve old retail information. Once the old retail information is retrieved then an array to insert into POS_MODS table is filled in the fill_pos_mods() function.

After the call to the fill_pos_mods function, the package TICKET_SQL.PRICE_CHANGE is called to create the appropriate records for printing tickets based on the price change.

process_style()

SKU components of style items are selected from rag_skus_st and process_sku() is called for each fetched sku.

fill_pos_mods()

The main task of this function is to fill elements of arrays with information that will be inserted into the POS_MODS table. In addition to filling the arrays, it will also resize the insert (incrementing the size with the commit_max_ctr on RESTART_CONTROL) arrays when the array indexes will be extended.

pos_mods_insert()

Insert into POS_MODS from the insert arrays. The base package inserts SKU and UPC information derived from the price change tables.

update_head_status()

PRICE_SUSP_HEAD's status of the last processed price change is updated to 'E' for extracted whenever this function is called. If the origin of that price change was group level (pc_origin on PRICE_SUSP_HEAD = 'G') then the price_criteria head record for that price change is set to 'E' for extracted.

item_on_clearance()

This function checks the clearance suspense and reset tables to see if the current sku is on clearance at the current store.

size_input_arrays()

This function sizes two different sets of arrays. The arrays into which the price change information is fetched are allocated memory. Also the array used to hold the PRICE_SUSP_DETAIL rowids whose status will be updated is sized here. The commit_max_ctr field, on the RESTART_CONTROL table, determines the size of the arrays.

psd_update()

The PRICE_SUSP_DETAIL table is updated, the status set to 'E', for all of the records whose rowids are in the update array.

I/O Specification

N/A

Retek Price Change Update [pcext]

Design Overview

The Price Change Extract (pcext) module selects permanent price change records, which are set to go into effect tomorrow and updates the RMS item/location and item/zone tables with the new prices. The price changes processed by this program include both price change events set up in the pricing module and price changes generated by a SKU's zone group changing. In addition to the updates of the retail prices in the system, records are also written for price history and transaction-level stock ledger. If a SKU/style price change results in a change to the price in the base zone, then the retail at all warehouse locations will also be updated. (Note: If the price change is for a style, then this price change also affects all of the SKUs within that style.) For SKU zone group changes, all stores are updated with the price from the base zone. For regular price changes the new retail is applied to the appropriate stores within the zone. Once this program has processed a price change, the status of the price change is changed to Delete.

Scheduling Constraints

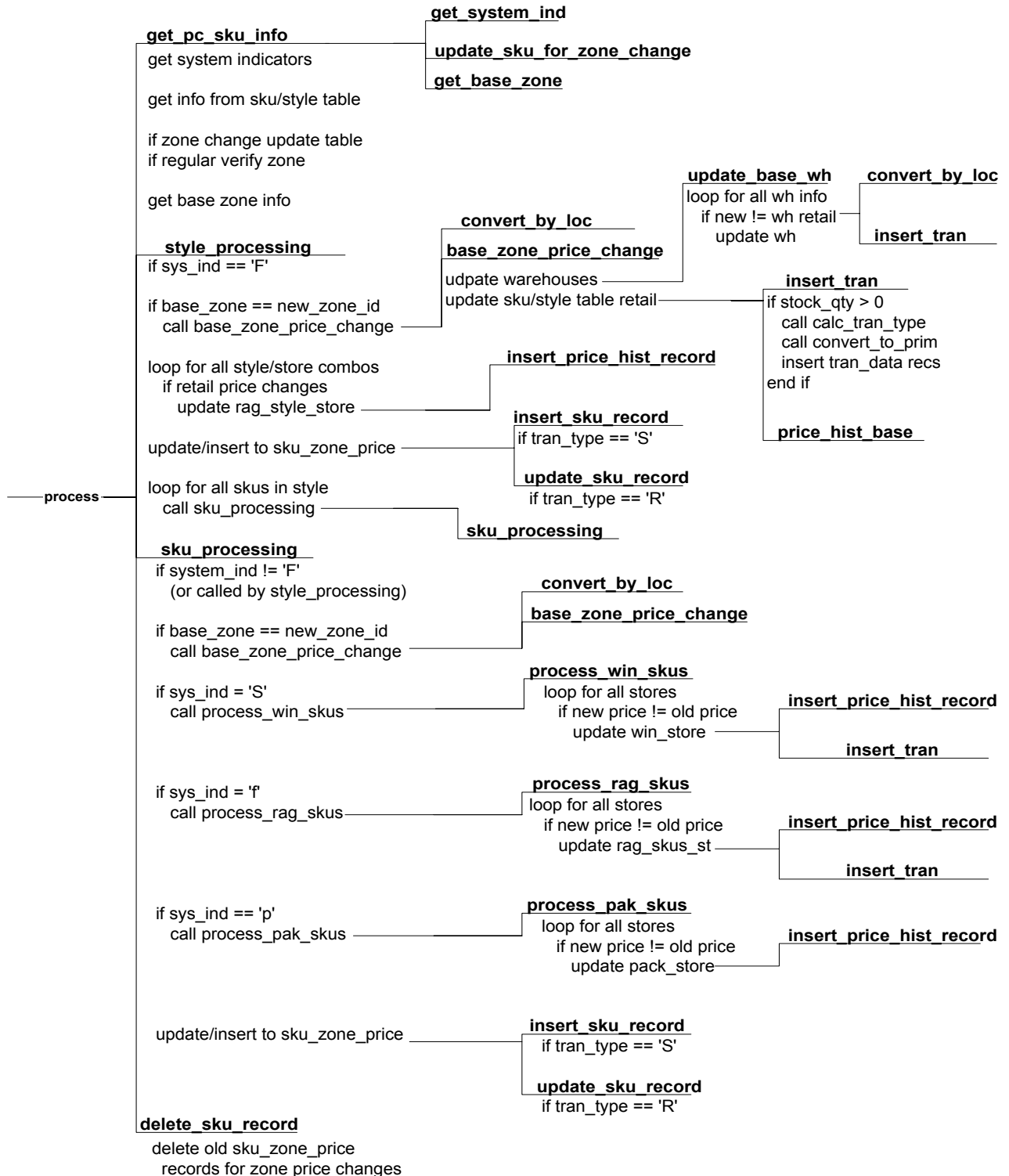
Processing Cycle:	Phase 3 - after pccrext
Scheduling Diagram:	No changes
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	The threading mechanism will be price change. The v_restart_price_chg view will be used.

Restart Recovery

The Logical Unit of Work for pcext is price change transaction type (Regular or Zone price changes), price change and sku. Processing is done as master-detail, however, price_susp_detail records are updated as they are processed and so commits can occur at any time during the processing (i.e. the commit process does not need to occur after an update to the header record).

Program Flow

PCEXT.PC Program Diagram



Shared Modules

CURRENCY_SQL.CONVERT - this package will be used to handle all currency conversions

GET_TRANSFER_QTY - procedure retrieves all stock quantities that are currently in transit

STKLEDGR_SQL.TRAN_DATA_INSERT - inserts records to tran_data table

PRCCHGTYP - determines type of price change (markup, markdown, etc.)

GET_SYSTEM_IND - retrieves system indicator variable

Function Level Description

init()

This function fetches the system date for tomorrow.

process()

This function will loop through the records retrieved by the driving cursor which should be ordered by price change tran_type and price change. Each SKU/style within a price change will be processed. First item information is gathered by the get_pc_sku_info function. If the price change item is a style then style processing is called, otherwise the sku_processing function will be called

get_pc_sku_info(). Once the item has been processed, the price_susp_detail record is updated to indicate that it has been processed. If the price change is a zone price change, then the sku_zone_price table is purged of records for that item that have the old zone information. Finally, if all of the items within the price change have been processed the price_susp_head record status is updated to processed.

get_pc_sku_info()

Generic SKU/style level information is gathered for the item. The get_system_ind function is called to get the system indicator variable. Retail, cost, zone, and supplier information is fetched from the sku/style and supplier tables. If processing a zone price change then the item table zone group is updated to the new zone group id. Finally, get_base_zone is called to determine the base zone for the item before returning to the process function.

get_system_ind()

System indicator information is retrieved using the GET_SYSTEM_IND stored procedure.

Update_sku_for_zone_change()

In the event of a zone price change, the win_skus or rag_style table are updated with the new zone group information.

Get_base_zone()

For regular price changes, the base zone can be retrieved from the sku_zone_price table, but for zone price changes the base zone must be retrieved from the price_zone table.

Style_processing()

This function will be called if the price change item is a fashion style. First it will be determined if the price change at hand is a base price change (i.e. the price change zone is equal to the base zone), if so, then the rag_style and rag_style_wh tables will be updated in the base_zone_price_change function. Next all style store locations are updated with the new retail price. If the retail is different then new price_hist records are inserted by the price_hist_insert_record function. Next the sku_zone_price table will have to be modified to reflect the new retail. Insert_sku_record is called if processing a zone price change, and update_sku_record is called in the event of a regular price change. Finally, sku_processing will be called to process price changes for each sku in the style.

Sku_processing()

Sku_processing can be called from two places: if the original price_change item is a sku then this function will be called directly from the process function, however, it will also be called from the style_processing function to process each sku within a style. The system indicator is passed as a parameter to this function to protect the original system indicator of the price change item. Like style_processing, base retail information will be updated by a call to base_zone_price_change if the price change zone is equal to the sku's base zone. Next sku/store records will be update by calling the function appropriate to the item's system indicator. Finally, the retail price is updated on sku_zone_price for regular price changes, and new records are inserted into sku_zone_price for zone price changes.

Delete_sku_record()

Sku_zone_price records referring to the old zone information are deleted by this function.

Convert_by_loc()

The CURRENCY_SQL.CONVERT package will convert currencies according to passed parameters. The intent of this function is to reduce the amount of code by allowing the package to be called generically from anywhere in pcext.

Base_zone_price_change()

This function is called when the price change going into effect is occurring in the base zone for the zone group. All warehouses/item records are updated with the new retail price by calling the update_base_wh function. When the item table retail is updated, insert_tran and pric_hist_base are called to insert tran_data and price_hist records respectively.

Update_base_wh()

Loop through all item/warehouse combinations. First convert the new unit retail to the warehouse currency by calling convert_by_loc. If the retail is different, update the item/warehouse table with the new retail and call insert_tran to insert tran_data records for the price change.

Price_hist_base()

Base price changes will call this function to add base price rows to the price_hist table.

Insert_price_hist_record()

Price history records are inserted according to the price change type (single/multi-currency, etc)

insert_sku_record()

New sku_zone_price records are inserted for the item with the new zone group and zone id information for zone price changes. Retail will remain at the local/zone currency.

update_sku_record()

Unit retail is updated according to the regular price change specifications. Retail will remain at the local/zone currency.

process_win_skus()

Staple skus are updated with the new retail price at each of the stores within the price zone. This function will use different cursors for selecting sku/store information depending on the type of price change being processed. For regular price changes the old retail information is selected from the sku_zone_price table where the zone group and zone id are equal to the new zone group and zone id. For zone price changes, the cursor must join to the sku_zone_price table twice; once to get all the stores in the new zone group/id combination, and the second time to find the old retail information associated with the stores selected but in the zone group/id combination to which they used to belong. Once the appropriate cursors have been opened and fetched, win_store will be updated with the new retail (if it has changed) and tran_data and price_hist records will be written by the insert_tran and insert_price_hist_record functions.

`process_rag_skus()`

Fashion skus are updated with the new retail price at each of the stores within the price zone. This function will use different cursors for selecting sku/store information depending on the type of price change being processed. For regular price changes the old retail information is selected from the `sku_zone_price` table where the zone group and zone id are equal to the new zone group and zone id. For zone price changes, the cursor must join to the `sku_zone_price` table twice; once to get all the stores in the new zone group/id combination, and the second time to find the old retail information associated with the stores selected but in the zone group/id combination to which they used to belong. Once the appropriate cursors have been opened and fetched, `rag_skus_st` will be updated with the new retail (if it has changed) and `tran_data` and `price_hist` records will be written by the `insert_tran` and `insert_price_hist_record` functions.

`process_pak_skus()`

Pack item are updated with the new retail price at each of the stores within the price zone. This function will use different cursors for selecting sku/store information depending on the type of price change being processed. For regular price changes the old retail information is selected from the `sku_zone_price` table where the zone group and zone id are equal to the new zone group and zone id. For zone price changes, the cursor must join to the `sku_zone_price` table twice; once to get all the stores in the new zone group/id combination, and the second time to find the old retail information associated with the stores selected but in the zone group/id combination to which they used to belong. Once the appropriate cursors have been opened and fetched, `packstore` will be updated with the new retail (if it has changed). Packstore price changes will generate price history records by calling `insert_price_hist_record`, however, data will not be inserted into `tran_data`.

`Insert_tran()`

This function is called every time the retail price is changed at a stock location. Any in-transit stock is fetched by the `GET_TRANSFER_QTY` stored procedure. If the total stock quantity (stock on hand + transfer quantity) is greater than zero then the transaction type (markdown, etc) is determined by the `calc_tran_type` function. The retail information is converted into the primary currency, and then `tran_data` (via the `STKLEDGR_SQL.TRAN_DATA_INSERT` package) and `sup_data` records are inserted.

`Cacl_tran_type()`

The `PRCCHGTYP` stored procedure is called to determine the `tran_data` type code.

`Convert_to_prim()`

This function converts the retail into the primary currency for stock ledger inserts.

I/O Specification

N/A

Technical Issues

N/A

Price Change Impact [pcimpc]

Design Overview

The Price Change Impact (pcimpc) program updates the impact quantity and impact amount fields on the price change header table to take into account the impact of the price change, in both dollars and units, based on the current stock on hand.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PRICE_SUSP_HEAD	Yes	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Ad Hoc

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: price_change

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

PCIMPACT_HEAD – This procedure does the actual calculations used to determine the impact of the price change

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Price Change Overlap [pcovrl]

Design Overview

The Price Change Overlap program searches through all submitted price change records and checks for conflicts with other price changes in submitted, approved or extracted status. Conflicts that are found in this program are written to a price change conflicts table, which can be used for reporting purposes. Each time this program is run, the price conflict table is cleared, so that all conflicts on the table are current conflicts.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PRICE_CONFLICT	No	No	No	No	Yes

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: N/A

Pre-Processing: The table price_conflict is truncated in pre-processing

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

PCOVERLAP_BATCH – This is the procedure that actually does all of the overlap checking.

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Promotion Overlap Analysis [pcovrlpq]

Design Overview

When submitting a promotion for approval, the event must be checked to ensure that none of the affected items are on promotion at the affected locations. This process can take an inordinate amount of time when performed online, so promotions needing analysis are queued up for later processing. pcovrlpq (Queued Promotion Overlap Analysis) cycles through submitted promotions, finds any overlaps with other promotions, and writes them to the PRICE_OVERLAP_LOG table for reporting.

Scheduling Constraints

Processing Cycle Ad-hoc; can run during the online day if needed.

Scheduling Diagram: This program has no functional dependencies, but it should avoid running concurrently with prmext.pc to reduce the risk of resource contention.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

N/A.

Program Flow

N/A.

Shared Modules

PRICING_OVERLAP_SQL.PROM_OVERLAP: PL/SQL stored procedure that actually does all the work of finding overlaps for a given promotion.

Function Level Descriptions

main

Logs on to the database, calls process() to handle the work, writes messages to the daily logs.

process

Fetches promotions from the PRICE_OVERLAP_QUEUE table and calls PR_overlap() for each one.

PR_overlap

Calls the PRICING_OVERLAP_SQL.PROM_OVERLAP stored procedure to find and log overlaps.

I/O Specification

N/A.

Technical Issues

N/A.

Purge Price Changes [pcprg]

Design Overview

The Price Change Purge program is responsible for removing old price changes from the system. Price changes are removed from the system using the following criteria:

- the status of the price changes is Canceled or Deleted
- the status is Extracted, but the effective date was yesterday
- the status of the price change is Rejected and the effective date of the price change has met the requirement for the number of days that rejected price changes are held.

Note: The number of days that rejected price changes are held is determined by a system option.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
PRICE_CONFLICT	No	No	No	No	Yes
PRICE_CHANGE_CONFLICT_LOG	No	No	No	No	Yes
PRICE_EVENT_ZONE	No	No	No	No	Yes
PRICE_SUSP_DETAIL	No	No	No	No	Yes
PRICE_SUSP_HEAD	No	Yes	No	No	Yes
UNIT_OPTIONS	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Ad Hoc

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

Re-run:

If this program terminates normally, the price change suspense tables must be recovered.

If this program terminates abnormally, restart without recovery.

Program Flow

N/A

Shared Modules

N/A

Function Level Description

All of the functionality of this program is performed in the process() function:

This function will select records from price_susp_head that should be deleted or purged from the system. This includes all cancelled or deleted records, all rejected records that are older than allowed according to the number_days_rejects_held field on the unit_options table, and all records that are in extracted status and have an activation date of yesterday.

The selected records are fetched into a loop where the deletes occur. Inside the while loop records are deleted from price_conflict, price_even_zone, price_susp_detail, price_change_conflict_log, and price_susp_head where the price_change on those tables is the same as the selected price change.

I/O Specification

N/A

Technical Issues

N/A

[pctrandn.pc]

Design Overview

The Price Transactions Download for Store Zone Changes (pctrandn) module performs the first step in processing pricing transactions that are created when a store is moved from one zone to another. This module will download the price changes generated from store zone change. It will select all items that exist at the store that will be moved and are associated with the zone group in which the move was made. Then, records are written out to the POS interface table, similar to permanent price changes, such that the store(s) affected will be notified.

Scheduling Constraints

Processing Cycle: Phase 3 - Nightly

Scheduling Diagram: Prior to running Pctranex.pc

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Store

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD		X			
PRICE_BATCH_TRAN		X			
WIN_SKUS	X	X			
RAG_SKUS	X	X			
RAG_STYLE		X			
WIN_STORE	X	X			
RAG_SKUS_ST	X	X			
PACKHEAD	X	X			
PACKSTORE	X	X			
POS_MOD			X		
ITEM_ZONE_PRICE	X	X			
PRICE_ZONE_GROUP_STORE	X	X			
SYSTEM_OPTIONS		X			

Restart Recovery

The restart string is new zone group + store + old zone group + item identifier. The restart is split between two cursors, with c_tran being the main cursor and one of c_win_skus, c_packhead, or c_rag_skus being used depending on whether the item being processed is a staple item, a pack item, or a fashion item. The commit counter should be set to 10,000 records.

Driving Cursor:

```
EXEC SQL DECLARE c_tran CURSOR FOR

SELECT pbt.zone_group_id,

       pbt.store,

       pbt.old_zone_id

FROM v_restart_store rv,

     price_batch_tran pbt

WHERE rv.driver_value = pbt.store

      AND rv.driver_name = :restart_driver_name

      AND rv.thread_val = :restart_thread_val

      AND rv.num_threads = :restart_num_threads

      AND (pbt.zone_group_id >= NVL(:restart_zone_group, -999) OR

           (pbt.zone_group_id = :restart_zone_group AND

            (pbt.store > :restart_store OR

             (pbt.store = :restart_store AND

              (pbt.old_zone_id >= :restart_zone_id))))))

      AND pbt.effective_date <= L_vdate + L_pos_ext_days

ORDER BY pbt.zone_group_id, pbt.store, pbt.old_zone_id;
```

Examples of Sku_processing cursors:

```
/* Cursor to retrieve records from win_skus table */
```

```
EXEC SQL DECLARE c_win_skus CURSOR FOR

SELECT w.sku,
```

```

        w.dept,
        w.class,
        w.subclass,
        NVL(s.clear_ind,'N'),
        DECODE(:os_std_av_ind,'S',
              NVL(s.unit_cost,0),NVL(s.av_cost,0)),
        s.stock_on_hand + in_transit_qty,
        ';' || TO_CHAR(:pbt_zone_group) ||
        ';' || TO_CHAR(:old_zone_id) ||
        ';' || TO_CHAR(:store) ||
        ';' || TO_CHAR(w.sku) ||
        ';' || 'S'
FROM win_skus w,
     win_store s
WHERE w.zone_group_id = :pbt_zone_group
AND s.sku = w.sku
AND s.store = :store
AND w.sku > :ps_restart_sku
ORDER BY w.sku;

```

/* Cursor to retrieve records from packhead table */

```
EXEC SQL DECLARE c_packhead CURSOR FOR
```

```

SELECT packhead.pack_no,
       packhead.dept,
       packhead.class,
       packhead.subclass,

```

```
        '||TO_CHAR(:pbt_zone_group)||
        '||TO_CHAR(:store)||
        '||TO_CHAR(:old_zone_id)||
        '||TO_CHAR(packhead.pack_no)||
        '||'P'

FROM packhead,

WHERE packhead.zone_group_id = :pbt_zone_group

      AND packhead.pack_no > :restart_sku

ORDER BY packhead.pack_no;

/* Cursor to get fashion SKU records*/

EXEC SQL DECLARE c_rag_skus CURSOR FOR

SELECT rsku.sku,

       rstyle.dept,

       rstyle.class,

       rstyle.subclass,

       NVL(rstore.clear_ind,'N'),

       DECODE(:os_std_av_ind,'S',

              NVL(rstore.unit_cost,0),NVL(rstore.av_cost,0)),

       rstore.stock_on_hand + in_transit_qty,

       '||TO_CHAR(:pbt_zone_group)||

       '||TO_CHAR(:old_zone_id)||

       '||TO_CHAR(:store)||

       '||TO_CHAR(rsku.sku)||

       '||'F'

FROM rag_skus rsku,
```

```

        rag_style rstyle,
        rag_skus_st rstore
WHERE rstore.sku = rsku.sku

AND rstore.store = :store

AND rsku.style = rstyle.style

AND rstyle.zone_group_id = :pbt_zone_group

AND rsku.sku > :ps_restart_sku

ORDER BY rsku.sku;

```

Program Flow

N/A

Shared Modules

POS_UPDATE_SQL.POS_MODS_INSERT – This procedure will be called to do the inserts into the POS interface table (pos_mods).

Function Level Description

init()

In addition to performing standard restart tasks, the current date (vdate) is selected from the period table and the POS_extract_days is selected from the system_options table.

process()

Price_batch_tran is the driving table for the program. Store Zone Changes are selected in this function based on the number of POS extract days and the effective date of the zone change. If today's date (vdate) is within the number of POS extract days from the zone change effective date, then the store information including new and old zone information will be selected in this function. If the zone change is within POS extract days of its effective date then call the sku_processing function for each item type ('S' staple, 'P' pack, and 'F' fashion). Each call will be made passing in the item type parameter. The sku_processing function will execute one of three cursors (depending on the item type) to retrieve all of the items in the zone group of the zone change.

create_pos_mods()

This function will be called by the sku_processing function and will perform inserts into pos_mods for a price change and also performs the tasks necessary to perform the insert. Sku information for pos_mods table inserts is fetched from desc_look. Old and new retail information is compared to determine the type of price change being sent to the point of sales. Finally, records are inserted into pos_mods for the sku by calling POS_UPDATE_SQL.POS_MODS_INSERT.

sku_processing()

This function will be called from the process function for each item type ('S', 'P', 'F') and for each zone change that must be downloaded to the stores. Information for the sku or pack item is now fetched and processed in this function by executing one of three cursors. One for each of the three item types. If the item exists at the current store and is not on clearance in the old zone then processing continues, otherwise skip to the next sku. The old and new retail information is fetched from item_zone_price based on the new and old zone information selected from price_batch_tran. If either the single or multi-unit price changes, then the create_pos_mods() function is called to insert pos_mods records for the POS.

final ()

This function will perform the standard restart/recovery close logic.

I/O Specification

None.

Technical Issues

None.

[pctranex.pc]

Design Overview

The Price Transactions Extract for Store Zone Changes module performs the final steps in processing pricing transactions that are created when a store is moved from one zone to another. This module will extract the price changes generated from store zone change that will become effective tomorrow and update the RMS tables to reflect the new prices. It will select all items that exist at the store that was moved and are associated with the zone group in which the move was made. Then, updates are made to the item/store prices, along with inserts into price history, supplier history, and transaction-level stock ledger tables.

Scheduling Constraints

Processing Cycle: Phase 3 - Daily

Scheduling Diagram: After the pctrandn.pc module is run

Pre-Processing: N/A

Post-Processing: Processed records will be deleted from the price_batch_tran table.

Threading Scheme: Store

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ITEM_SUPPLIER	X	X			
PERIOD		X			
PRICE_BATCH_TRAN		X			
WIN_SKUS	X	X			
RAG_SKUS	X	X			
RAG_STYLE		X			
WIN_STORE	X	X		X	
RAG_SKUS_ST	X	X		X	
PACKHEAD	X	X			
PACKSTORE	X	X		X	
PRICE_HIST			X		
PRICE_ZONE	X				X
ITEM_ZONE_PRICE	X	X			X

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PRICE_ZONE_GROUP_STORE	X	X		X	
SUP_DATA			X		
SYSTEM_OPTIONS		X			

Restart Recovery

The restart string is new zone + store + old zone + item identifier. The restart is split between two cursors, with c_tran being the main cursor and one of c_win_skus, c_packhead, or c_rag_skus being used depending on whether the item being processed is a staple item, a pack item, or a fashion item. The commit counter should be set to 10,000 records.

Driving Cursor:

```
EXEC SQL DECLARE c_tran CURSOR FOR

SELECT pbt.zone_group_id,

       pbt.store,

       pbt.old_zone_id

FROM v_restart_store rv,

     price_batch_tran pbt

WHERE rv.driver_value = pbt.store

      AND rv.driver_name = :restart_driver_name

      AND rv.thread_val = :restart_thread_val

      AND rv.num_threads = :restart_num_threads

      AND (pbt.zone_group_id >= NVL(:restart_zone_group, -999) OR

           (pbt.zone_group_id = :restart_zone_group AND

            (pbt.store > :restart_store OR

             (pbt.store = :restart_store AND

              (pbt.old_zone_id >= :restart_zone_id))))))

      AND pbt.effective_date = L_vdate + 1

ORDER BY pbt.zone_group_id, pbt.store, pbt.old_zone_id;
```

Examples of cursors in the Sku_processing function:

```
/* Cursor to retrieve records from win_skus table */
```

```
EXEC SQL DECLARE c_win_skus CURSOR FOR
```

```
    SELECT win_skus.sku,  
           win_skus.dept,  
           win_skus.class,  
           win_skus.subclass,  
           ';' || TO_CHAR(:pbt_zone_group) ||  
           ';' || TO_CHAR(:store) ||  
           ';' || TO_CHAR(:old_zone_id) ||  
           ';' || TO_CHAR(win_skus.sku) ||  
           ';' || 'S'
```

```
    FROM win_skus,
```

```
           item_supplier
```

```
    WHERE win_skus.zone_group_id = :pbt_zone_group
```

```
          AND item_supplier.item = win_skus.sku
```

```
          AND item_supplier.primary_supp_ind = 'Y'
```

```
          AND win_skus.sku > :restart_sku
```

```
    ORDER BY win_skus.sku;
```

```
/* Cursor to retrieve records from packhead table */
```

```
EXEC SQL DECLARE c_packhead CURSOR FOR
```

```
    SELECT packhead.pack_no,  
           packhead.orderable_ind  
           packhead.dept,  
           packhead.class,
```

```
        packhead.subclass,  
        '||TO_CHAR(:pbt_zone_group)||  
        '||TO_CHAR(:store)||  
        '||TO_CHAR(:old_zone_id)||  
        '||TO_CHAR(packhead.pack_no)||  
        '||'P'  
FROM packhead,  
        item_supplier  
WHERE packhead.zone_group_id = :pbt_zone_group  
        AND item_supplier.item = packhead.pack_no (+)  
        AND item_supplier.primary_supp_ind = 'Y' (+)  
        AND packhead.pack_no > :restart_sku  
ORDER BY packhead.pack_no;
```

```
/* Cursor to get fashion SKU records*/
```

```
EXEC SQL DECLARE c_rag_skus CURSOR FOR
```

```
SELECT rsku.sku,  
        rstyle.dept,  
        rstyle.class,  
        rstyle.subclass,  
        TO_CHAR(item_supplier.supplier),  
        '||TO_CHAR(:pbt_zone_group)||  
        '||TO_CHAR(:store)||  
        '||TO_CHAR(:old_zone_id)||  
        '||TO_CHAR(rsku.sku)||  
        '||'F'
```

```
FROM rag_skus rsku,  
     rag_style rstyle,  
     item_supplier  
WHERE rstyle.zone_group_id = :pbt_zone_group  
AND item_supplier.item = rsku.sku  
AND item_supplier.primary_supp_ind = 'Y'  
AND rsku.style = rstyle.style  
AND rsku.sku > :restart_sku  
ORDER BY rsku.sku;
```

Program Flow

N/A

Shared Modules

PRCCHGTYP – Calculates the tran_type for the tran_data insert.

STKLEDGR_SQL.TRAN_DATA_INSERT – Inserts a record into tran_data.

SUPP_ITEM_SQL.GET_PRI_SUP_COST – Gets the primary supplier cost for an item.

ITEMLOC_QTY_SQL.GET_TRANSFER – Determines stock amounts in transit (for tran_data insert).

Function Level Description

init()

In addition to performing standard restart tasks, the current date is selected from the period table, and the std_av_ind is selected from the system_options table to be used in selecting the correct unit_cost in the sku_processing function for insert into the price_hist table in the create_price_hist function.

process()

Price_batch_tran is the driving table for the program. Store Zone Changes are selected in this function based on the effective date of the zone change. If today's date (vdate) is the day before the effective date, then the store information including new and old zone information will be selected in this function. If the zone change effective date is tomorrow then call the update_new_zone function to actually move the store to the new zone and the delete_old_zone function is called, which will delete the old zone information.. Next, call the sku_processing function for each item type ('S' staple, 'P' pack, and 'F' fashion) and for each zone change that will be effective tomorrow. Each call will be made passing in the item type parameter. The sku_processing function will execute one of three cursors (depending on the item type) to retrieve all of the items in the zone group of the zone change.

create_price_hist()

This function is called from the sku_processing function and inserts a record into price_hist. A different transaction type will be inserted depending on the type of price that changed: 4 for single unit price changes only, 10 for multi-unit price changes only, and 11 for single and multi-unit price changes.

sku_processing()

This function is called from the process function. Information for the sku or pack item is now fetched and processed in this function. If the item exists at the current store processing continues. If it does not, skip to the next sku. The old and new retail information is fetched from item_zone_price based on the new and old zone information selected from the price_batch_tran table. If the single unit retail price changes then the win_store, packstore, or rag_skus_st retail information is updated. If either the single or multi-unit price changes and the stock_qty is greater than zero (which will not occur for packs), then insert_tran() is called to insert records into tran_data, and create_price_hist is called to insert price_hist records. If neither retail changes then the sku should be skipped, since it is not changing in price at the store when the store moves zones.

delete_old_zone()

This function is called from the process function and will delete rows from zone tables where stores no longer exist in a zone_id. First the price_zone_group_store table is queried for all the stores in the old zone_id. If there are no records then rows with the old zone id/zone group are deleted from the appropriate zone tables: item_zone_price and price_zone.

update_new_zone()

This function is also called from the process function and will update the price_zone_group_store table for each store changing zones, setting the zone id equal to the new zone id on the price_batch_tran table.

calc_tran_type()

This function is called from the sku_processing function and calls the stored procedure prcchgtype to determine the tran_type inserted into the tran_data table.

insert_tran()

First this function calls calc_tran_type() to determine the type of tran_data insert. The store's stock on hand the quantity in transit must be evaluated to determine the stock_qty, so the ITEMLOC_QTY_SQL.GET_TRANSFER stored procedure is called. If the stock_qty is positive the STKLEDGR_SQL.TRAN_DATA_INSERT package is called to insert the tran_data record. If the price change is a markup or markdown cancellation then supplier information is inserted into sup_data with a transaction type of 10 (markdown at retail). Because the transaction type is a markdown the markup amount should be multiplied to -1 to get the appropriate sign for the insert (to effectively create a markup). A similar process occurs for the second set of tran_types and tran_type amounts (for multi_unit information) to insert tran_data and sup_data records.

final ()

This function will perform the standard restart/recovery close logic.

I/O Specification

None.

Technical Issues

None.

Class Level Planned Sales Units Acceptance [plncupld]

Design Overview

The purpose of this batch module is to accept new planned sales units at a class/store/end-of-week level from an outside planning system. This data is then inserted into or used to update the planned sales data on the CLASS_SALES_HIST table.

Detail processing is very limited. If there is not a row on the table to update, a new one is inserted with department, class, store, end-of-week date, and planned sales amount taken directly from the input file. The date procedures CAL_TO_454 and CAL_TO_454_HALF are used to calculate the half, month and week numbers to be inserted with that row. If the table is being updated, the old planned sales amount is overwritten with the planned sales amount extracted from the input file. Records with an end-of-week date before the current week's end will be rejected since only current and future plan numbers will be accepted into Retek.

Scheduling Constraints

Processing Cycle: At user discretion.

Scheduling Diagram: This program is independent of all other batch processing and can be run ad hoc.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

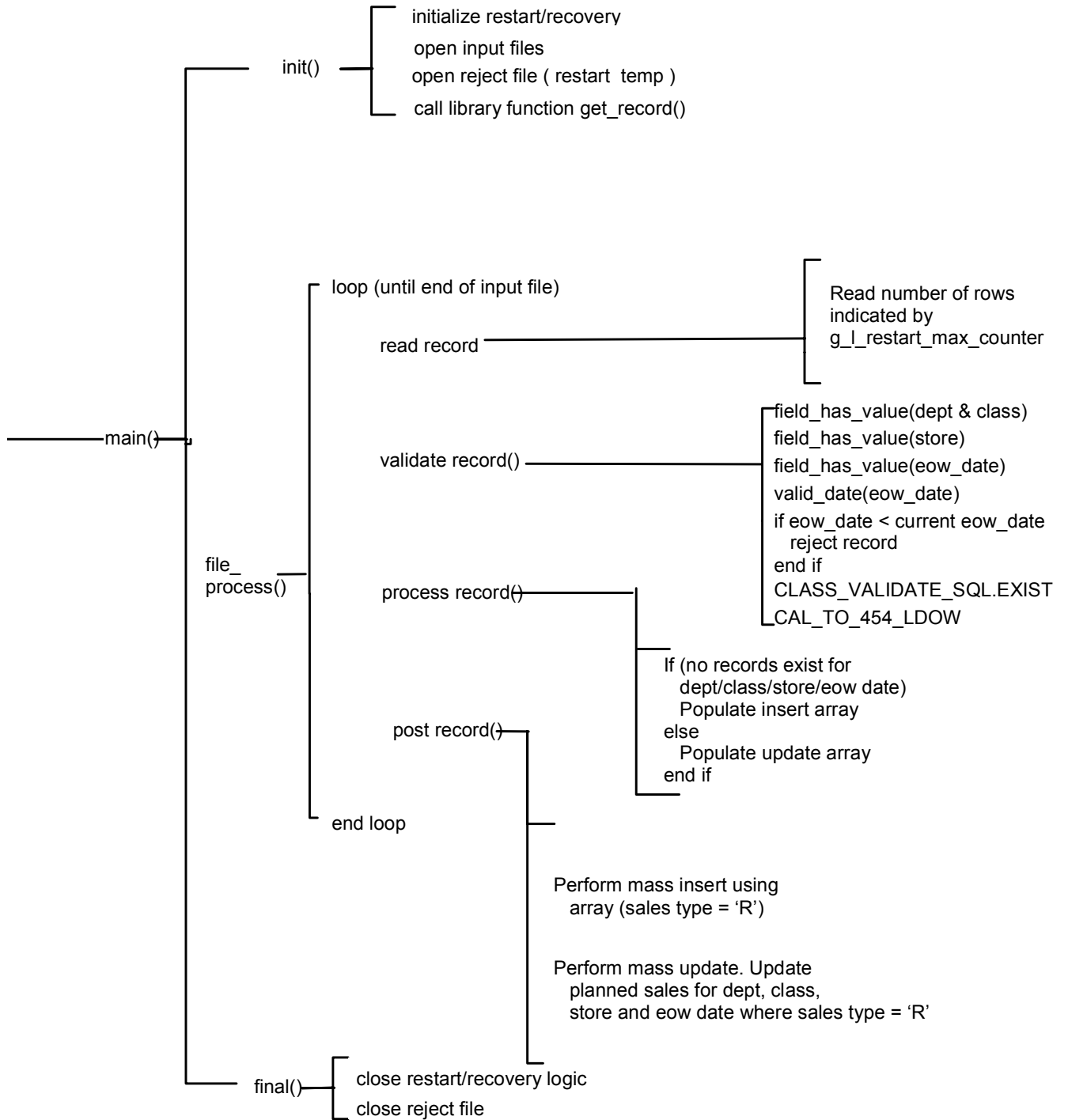
Restart Recovery

The logical unit of work for the planned sales import module is the discrete planned sales transaction. If a fatal error forces a restart, the restart will occur from the point in the file where the last commit occurred.

Error handling will also be based on this logical unit of work. If a record fails validation, it will be written to a rejected record file. This file will facilitate easy reprocessing once the error is fixed by writing the record exactly as it was in the source file. All errors will be written to an error log file using the WRITE_ERROR library function.

The recommended commit counter setting is 10,000. If a fatal error occurs and restart is necessary, processing will restart at the last commit point.

Program Flow



Shared Modules

`CLASS_VALIDATE_SQL.EXIST`: This packaged function will validate that a department/class combination exists on the class table.

`CAL_TO_454_LDOW`: This procedure will return an end-of-week date that can be used to validate the input record's end-of-week date.

Function Level Description

`init()`:

- Open files for input and output.
- Initialize restart/recovery.

`file_process()`:

- Read a record from the input file.
- Determine if the record is valid by calling the `validate_record()` function.
- If the record is valid, call the `process_record()` function to populate an insert or update array.
- Save the records to the `CLASS_SALES_HIST` table by calling the `post_record()` function.

`validate_record()`:

- Ensure that the department, class, store and end-of-week date fields all contain values by calling the function `field_has_value()` for each.
- Check that the input record's end-of-week date is a valid date in the correct format by calling the `valid_date()` function. If so, check that it is also a valid end-of-week date using `CAL_TO_454_LDOW`.
- Check that the input record's end-of-week date is greater than or equal to the current week's end date. Historic planned sales will not be interfaced into Retek. If the end-of-week date is before the current week's end, reject the record.
- Call the `CLASS_VALIDATE_SQL.EXIST` function to validate that the department/class combination exists on the `CLASS` table.

process_record():

- If no row exists on the CLASS_SALES_HIST table for the input record's department, class, store and end-of-week date, populate an array to insert a new one. Otherwise, populate an array to update the existing row.

post_record():

- Perform a mass update and/or insert using the two array structures. For the mass insert, the sales type should always be inserted with an 'R' (for regular sales). The mass update should update the planned sales amount for the input record's department, class, subclass, store and end-of-week date]where the sales type is equal to 'R'.

final():

- Close restart/recovery logic.
- Close the reject file.

on FATAL ERROR:

- Use standard batch error handling to write to the error log file.
- Exit the program.

on NON-FATAL ERROR:

- Use standard batch error handling to write to the error log file.
- Increment g_l_rej_cnt.

I/O Specification

Input File:

The input file name should be accepted as a runtime parameter at the command line.

If multi-threading will be used, the file name must contain a suffix (e.g. filename####) to identify the thread value so that multiple input files can be used concurrently.

The file will be formatted as fixed-width values with no delimiters or punctuation. A new-line character will indicate the end of each record.

The end-of-week date must be passed in the format 'YYYYMMDDHHMISS' in order to validate it using valid_date().

A sample detail record would look like the following:

FDETL12345678901234567890123412341234123419980101123456789012

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	Char(10)	Line number in file	Unique line ID
	File Type Definition	Char(4)	PLCI	Identifies file as 'Planned Class Sales Import'
	File Create Date	Char(14)	create date	Date file was written by external system (formatted as 'YYYYMMDDHHM ISS')
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Char(10)	Line number in file	Unique line ID
	Transaction Set Control Number	Char(14)	specified by external system	Unique ID
	Department Identifier	Char(4)	numeric dept identifier	The internal Retek id number of a department - numeric character
	Class Identifier	Char(4)	numeric class identifier	The internal Retek id number of a class within the department given - numeric character
	Store Identifier	Char(4)	numeric store identifier	The internal Retek id number of a store - numeric character
	Eow Date	Char(14)	end of week date	The end of week date for the budgeted week (formatted as 'YYYYMMDDHHM ISS')

Record Name	Field Name	Field Type	Default Value	Description
	Planned Sales Units	Number(12) 4 implied decimals	Planned Sales Units	Planned sales units for the specified dept/class/store/week
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Char(10)	Line number in file	Unique line ID
	Control Number File Line Count	Number(10)	total detail lines	Sum of all transaction lines, not including file header and trailer

Reject File:

The reject file will be formatted identically to the input file for easy re-processing. Library functions will create the header and trailer records as well as keeping track of the rejected record count. A reject file will be created in all cases. If no errors occur, the reject file will consist only of file header and trailer records, and the file line count will be equal to 0. The reject file name should be specified as a runtime parameter.

Error File:

Standard Retek batch error handling will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

This program will read from an input flat file. It will use restart/recovery, array processing and will commit to the database at a frequency determined by the `commit_max_ctr` on the `RETEK_CONTROL` table.

Department Level Planned Sales Units Acceptance [plndupld]

Design Overview

The purpose of this batch module is to accept new planned sales units at a department/store/end-of-week level from an outside planning system. This data is then inserted into or used to update the planned sales data on the DEPT_SALES_HIST table.

Detail processing is very limited. If there is not currently a row on the table to update, a new row is inserted with department, store, end-of-week date and planned sales amount taken from the input file. The date procedures CAL_TO_454 and CAL_TO_454_HALF are used to calculate the half, month and week numbers to be inserted with that row. If the table is being updated, the old planned sales amount is overwritten with the planned sales amount extracted from the input file. Records with an end-of-week date before the current week's end will be rejected since only current and future plan numbers will be accepted into Retek.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DEPT_SALES_HIST	No	No	Yes	Yes	No
DEPT	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: At user discretion.

Scheduling Diagram: This program is independent of all other batch processing and can be run ad hoc.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

The logical unit of work for the planned sales import module is the discrete planned sales transaction. If a fatal error forces a restart, the restart will occur from the point in the file where the last commit occurred

Error handling will also be based on this logical unit of work. If a record fails validation, it will be written to a rejected record file. This file will facilitate easy reprocessing once the error is fixed by writing the record exactly as it was in the source file. All errors will be written to an error log file using the `WRITE_ERROR` library function.

The recommended commit counter setting is 10,000. If a fatal error occurs and restart is necessary, processing will restart at the last commit point.

Shared Modules

`DEPT_VALIDATE_SQL.EXIST`: This packaged function will validate that a department exists on the `DEPS` table.

`CAL_TO_454_LDOW`: This procedure will return an end-of-week date that can be used to validate the input record's end-of-week date.

Function Level Description

`init()`:

Open files for input and output.

Initialize restart/recovery.

`file_process()`:

Read a record from the input file.

Determine if the record is valid by calling the `validate_record()` function.

If the record is valid, call the `process_record()` function to populate an insert or update array.

Save the records to the `DEPT_SALES_HIST` table by calling the `post_record()` function.

`validate_record()`:

Ensure that the department, store and end-of-week date fields all contain values by calling `field_has_value()` function for each.

Check that the input record's end-of-week date is a valid date in the correct format by calling the `valid_date()` function. If so, check that it is also a valid end-of-week date using `CAL_TO_454_LDOW`.

Check that the input record's end-of-week date is greater than or equal to the current week's end date. Historic planned sales will not be interfaced into Retek. If the input record's end-of-week date is before the current week's end, reject the record.

Call the `DEPT_VALIDATE_SQL.EXIST` function to validate that the record's department exists on the department table.

`process_record()`:

If no record exists on the `DEPT_SALES_HIST` table for the input record's department, store and end-of-week date, populate an array to insert a new one. Otherwise, populate an array to update the existing row.

`post_record()`:

Perform a mass update and/or insert using the two array structures. For the mass insert, the sales type should always be inserted with an 'R' (for regular sales). The mass update should update the planned sales amount for the input record's department, store and end-of-week date where the sales type is equal to 'R'.

`final()`:

Close restart/recovery logic.

Close the reject file.

on FATAL ERROR:

Use standard batch error handling to write to the error log file.

Exit the program.

on NON-FATAL ERROR:

Use standard batch error handling to write to the error log file.

Increment `g_l_rej_cnt`.

I/O Specification

Input File:

The input file name should be accepted as a runtime parameter at the command line.

If multi-threading will be used, the file name must contain a suffix (i.e. filename####) to identify the thread value so that multiple input files can be used concurrently.

The file will be formatted as fixed-width values with no delimiters or punctuation. A new-line character will indicate the end of each record.

The end-of-week date must be passed in the format 'YYYYMMDDHHMISS' in order to validate it using valid_date().

A sample detail record would look like the following:

FDETL1234567890123456789012341234123419980101050000123456789012

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	Char(10)	Line number in file	Unique line ID
	File Type Definition	Char(4)	PLDI	Identifies file as 'Planned Department Sales Import'
	File Create Date	Char(14)	create date	Date file was written by external system (formatted as 'YYYYMMDDHHMISS')
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Char(10)	Line number in file	Unique line ID
	Transaction Set Control Identifier	Char(14)	specified by external system	Unique ID
	Department Identifier	Char(4)	numeric dept identifier	The internal Retek id number of a department - numeric character
	Store Identifier	Char(4)	numeric store identifier	The internal Retek id number of a store - numeric character

Record Name	Field Name	Field Type	Default Value	Description
	Eow Date	Char(14)	end of week date	The end of week date for the budgeted week (formatted as 'YYYYMMDDHHMISS')
	Planned Sales Units	Number(12) 4 implied decimals	Planned Sales Units	Planned sales units for the specified dept/store/week
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Char(10)	Line number in file	Unique line ID
	Control Number File Line Count	Number(10)	total detail lines	sum of all transaction lines, not including file header and trailer

Reject File:

The reject file will be formatted identically to the input file for easy re-processing. Library functions will create the header and trailer records as well as keeping track of the rejected record count. A reject file will be created in all cases. If no errors occur, the reject file will consist only of file header and trailer records, and the file line count will be equal to 0. The reject file name should be specified as a runtime parameter.

Error File:

Standard Retek batch error handling will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

This program will read from an input flat file. It will use restart/recovery, array processing and will commit to the database at a frequency determined by the commit_max_ctr on the RETEK_CONTROL table.

Subclass Level Planned Sales Units Acceptance [plnsupld]

Design Overview

The purpose of this batch module is to accept new planned sales units at a subclass/store/end-of-week level from an outside planning system. This data is then inserted into or used to update the planned sales data on the SUBCLASS_SALES_HIST table.

Detail processing is very limited. If there is not currently a row on the table to update, a new row is inserted with the department, class, subclass, store, end-of-week date, and planned sales amount taken directly from the file. The date procedures CAL_TO_454 and CAL_TO_454_HALF are used to calculate the half, month and week numbers to be inserted with that row. If the table is being updated, the old planned sales amount is overwritten with the planned sales amount extracted from the import file. Records with an end-of-week date before the current week's end will be rejected since only current and future plan numbers will be accepted into Retek.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
SUBCLASS_SALES_HIST	No	No	Yes	Yes	No
SUBCLASS	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: At user discretion.

Scheduling Diagram: This program is independent of all other batch processing and can be run ad hoc.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

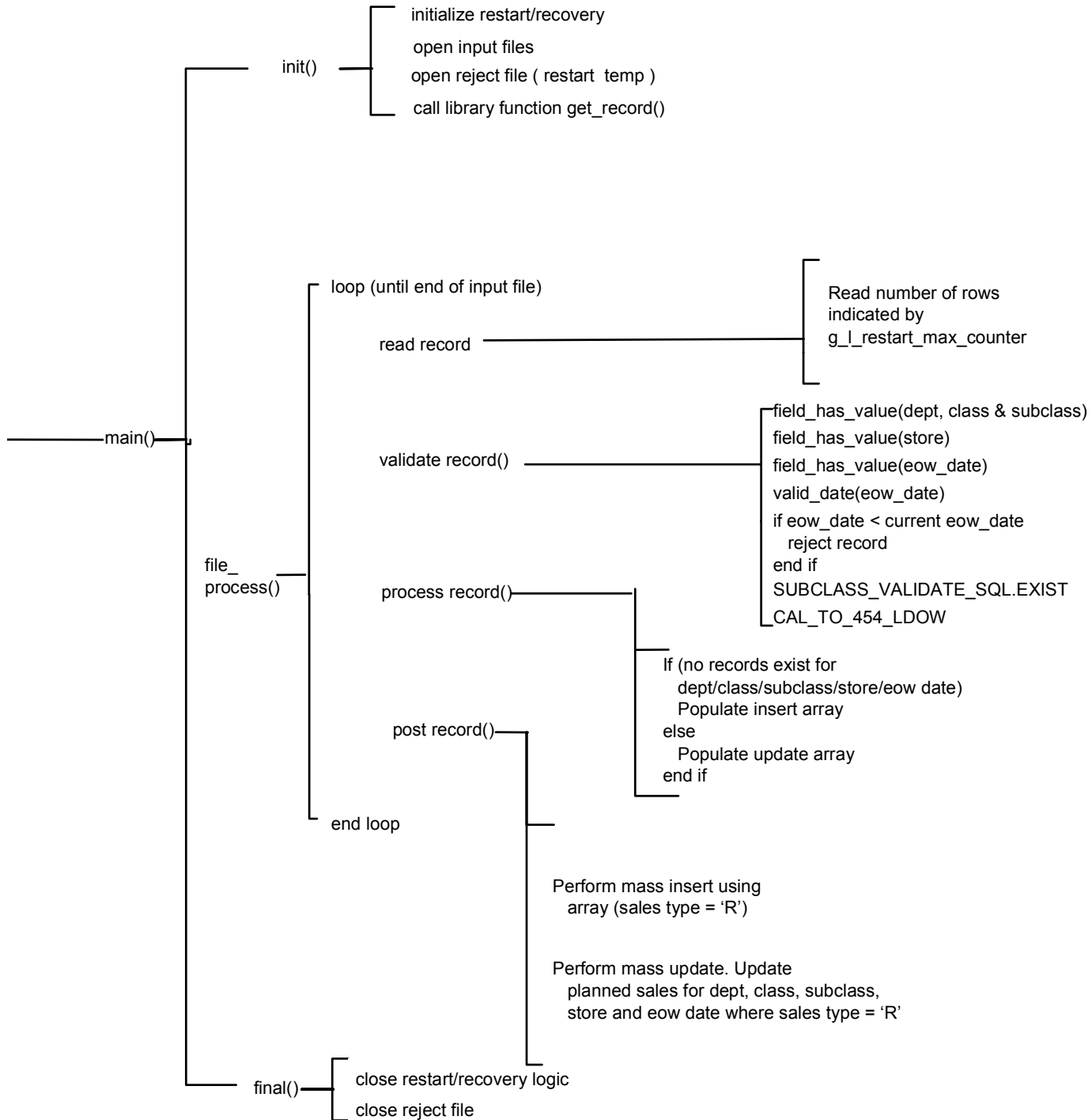
Restart Recovery

The logical unit of work for the planned sales import module is the discrete planned sales transaction. If a fatal error forces a restart, the restart will occur from the point in the file where the last commit occurred

Error handling will also be based on this logical unit of work. If a record fails validation, it will be written to a rejected record file. This file will facilitate easy reprocessing once the error is fixed by writing the record exactly as it was in the source file. All errors will be written to an error log file using the WRITE_ERROR library function.

The recommended commit counter setting is 10,000. If a fatal error occurs and restart is necessary, processing will restart at the last commit point.

Program Flow



Shared Modules

SUBCLASS_VALIDATE_SQL.EXIST: This packaged function will validate that a department/class/subclass combination exists on the subclass table.

CAL_TO_454_LDOW: This procedure will return an end-of-week date that can be used to validate the input record's end-of-week date.

Function Level Description

init():

- Open files for input and output.
- Initialize restart/recovery.

file_process():

- Read a record from the input file.
- Determine if the record is valid by calling the `validate_record()` function.
- If the record is valid, call the `process_record()` function to populate an insert or update array.
- Save the records to the `SUBCLASS_SALES_HIST` table by calling the `post_record()` function.

validate_record():

- Ensure that the department, class, subclass, store and end-of-week date fields all contain values by calling the function `field_has_value()` for each.
- Check that the input record's end-of-week date is a valid date in the correct format by calling the `valid_date()` function. If so, check that it is also a valid end-of-week date by using `CAL_TO_454_LDOW`.
- Check that the input record's end-of-week date is greater than or equal to the current week's end date. Historic planned sales will not be interfaced into Retek. If the input record's end-of-week date is less than the current week's end, reject the record.
- Call the `SUBCLASS_VALIDATE_SQL.EXIST` function to validate that the input record's department/class/subclass combination exists on the `SUBCLASS` table.

process_record():

- If no row exists on the SUBCLASS_SALES_HIST table for the input record's department, class, subclass, store and end-of-week date, populate an array to insert a new one. Otherwise, populate an array to update the existing row.

post_record():

- Perform a mass update and/or insert using the two array structures. For the mass insert, the sales type should always be inserted with an 'R' (for regular sales). The mass update should update the planned sales amount for the input record's department, class, subclass, store and end-of-week date where the sales type is equal to 'R'.

final():

- Close restart/recovery logic.
- Close the reject file.

on FATAL ERROR:

- Use standard batch error handling to write to the error log file.
- Exit the program.

on NON-FATAL ERROR

- Use standard batch error handling to write to the error log file.
- Increment g_l_rej_cnt.

I/O Specification

Input File:

The input file name should be accepted as a runtime parameter at the command line.

If multi-threading will be used, the file name must contain a suffix (i.e. filename###) to identify the thread value so that multiple input files can be used concurrently.

The file will be formatted as fixed-width values with no delimiters or punctuation. A new-line character will indicate the end of each record.

The end-of-week date must be passed in the format 'YYYYMMDDHHMISS' in order to validate it using valid_date().

A sample detail record would look like the following:

FDETL1234567890123456789012341234123412341998010150000123456789012

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	Char(10)	Line number in file	Unique line ID
	File Type Definition	Char(4)	PLSI	Identifies file as 'Planned Subclass Sales Import'
	File Create Date	Char(14)	create date	Date file was written by external system (formatted as 'YYYYMMDDHHMISS')
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Char(10)	Line number in file	Unique line ID
	Transaction Set Control Number	Char(14)	specified by external system	Unique ID
	Department Identifier	Char(4)	numeric dept identifier	The internal Retek id number of a department - numeric character
	Class Identifier	Char(4)	numeric class identifier	The internal Retek id number of a class within the department given - numeric character
	Subclass Identifier	Char(4)	numeric subclass identifier	The internal Retek id number of a subclass within the class given - numeric character
	Store Identifier	Char(4)	numeric store identifier	The internal Retek id number of a store - numeric character
	Eow Date	Char(14)	end of week date	The end of week date for the budgeted week (formatted as 'YYYYMMDDHHMISS')

Record Name	Field Name	Field Type	Default Value	Description
	Planned Sales Units	Number(12) 4 implied decimals	Planned Sales Units	Planned sales units for the specified dept/class/subclass/store/week
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Char(10)	Line number in file	Unique line ID
	Control Number File Line Count	Number(10)	total detail lines	Sum of all transaction lines, not including file header and trailer

Reject File:

The reject file will be formatted identically to the input file for easy re-processing. Library functions will create the header and trailer records as well as keeping track of the rejected record count. A reject file will be created in all cases. If no errors occur, the reject file will consist only of file header and trailer records, and the file line count will be equal to 0. The reject file name should be specified as a runtime parameter.

Error File:

Standard Retek batch error handling will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

This program will read from an input flat file. It will use restart/recovery, array processing and will commit to the database at a frequency determined by the commit_max_ctr on the RETEK_CONTROL table.

POS Configuration Download [Poscdnld.pc]

Design Overview

This program will handle the download of all Retek 'POS configurations' from the RMS database to a flat file for loading into a POS/Back Office application. The program will utilize a series of table statuses to identify delta records to send down on a nightly basis, thus keeping interface volume down.

Currently there are 7 identified POS configurations that will be handled by this program. They are: Coupons, Product Restrictions, Money Orders, Tender Types, Touch Buttons, Paid In-Out, and Supplier Payment Options at the Store. Of these 7 POS configurations, 5 contain data at the store level only, while 2 contain data at the item/store level. Furthermore, for the 2 POS configurations that are item/store specific, the items are associated with the POS configurations based on user-entered merchandise criteria. For example, a coupon configuration may apply to all items in departments 100, 200, and 300. The system does not store each coupon/item relationship, but rather the merchandise criteria (dept 100, dept 200, dept 300).

In all cases except one, delta records will be sent down nightly for POS configurations and only records whose effective dates are the next day will be sent down. Therefore, if a store is added to a POS configuration, only the new store would receive the new configuration and not all stores associated with it. The one exception is changes to the merchandise criteria. Once a merchandise configuration is modified, all items that meet the new criteria will be sent down (not the deltas). The overlay of items will be sent to the POS because RMS will not save the items that belong to the criteria, only the criteria itself. By saving criteria instead of criteria and all the SKUs that meet the criteria, there will be a substantial savings in disk space.

In addition to being responsible for downloading data to the stores, this program will handle the delete logic for the POS configurations. If a POS configuration, store (on pos_store), or merchandise criteria is set to delete status online, this program will send the delete notification to the store and purge the record(s) from the RMS database.

A post process will be run in the prepost.pc batch module to reset the status fields on money_ord_head, pos_coupon_head, pos_prod_rest_head, pos_sup_pay_criteria, pos_button_head, pos_tender_type_head, pos_payinout_head, pos_merch_criteria, and pos_store to NULL and set the extract_req_ind = 'N'.

Scheduling Constraints

Processing Cycle:	4
Scheduling:	This program should be run directly after posdnld.pc.
Pre-Processing:	None
Post-Processing:	Run prepost.pc to set statuses back to NULL.
Threading Scheme:	None

Restart Recovery

Logical unit of work

Pos_config_id and Pos_config_type

Driving Cursor:

Select payinout_id,

'PYIO',

pos_config_status,

ROWIDTOCHAR(rowid),

'||to_char(payinout_id)||';'||'PYIO'

From pos_payinout_head

Where extract_req_ind = 'Y'

And effective_date = TO_DATE(:ps_tomorrow,'YYYYMMDD')

And ((NVL(:ps_restart_pos_config_type,'PYIO') < 'PYIO'

OR(NVL(:ps_restart_pos_config_type,'PYIO') = 'PYIO'

AND payinout_id > :ps_restart_pos_config_id))

UNION ALL

Select tender_type_id,

'TEND',

pos_config_id,

```

ROWIDTOCHAR(rowid),
',' || to_char(tender_type_id) || ',' || 'TTYP'
From pos_tender_type_head
Where extract_req_ind = 'Y'
And effective_date = TO_DATE(:ps_tomorrow,'YYYYMMDD')
And ((NVL(:ps_restart_pos_config_type,'TTYP') < 'TTYP'
      OR(NVL(:ps_restart_pos_config_type,' TTYP') = 'TTYP'
      AND tender_type_id > :ps_restart_pos_config_id))
UNION ALL
Select button_conig_id,
'BTTN',
pos_config_id,
ROWIDTOCHAR(rowid),
',' || to_char(button_config_id) || ',' || 'BTTN'
From pos_button_head
Where extract_req_ind = 'Y'
And effective_date = TO_DATE(:ps_tomorrow,'YYYYMMDD')
And ((NVL(:ps_restart_pos_config_type,' BTTN') < 'BTTN'
      OR(NVL(:ps_restart_pos_config_type' BTTN') = 'BTTN'
      AND button_config_id > :ps_restart_pos_config_id))
UNION ALL
Select sup_pay_id,
'SUPP',
pos_config_status,
ROWODTOCHAR(rowid),
',' || to_char(sup_pay_id) || ',' || 'BTTN'

```

```
From pos_sup_pay_criteria

Where extract_req_ind = 'Y'

And effective_date = TO_DATE(:ps_tomorrow,'YYYYMMDD')

And ((NVL(:ps_restart_pos_config_type,' BTTN') < 'BTTN'

      OR(NVL(:ps_restart_pos_config_type,' BTTN') = 'BTTN'

      AND button_config_id > :ps_restart_pos_config_id))

UNION ALL

Select pos_prod_rest_id,

'PRST',

pos_config_status,

ROWIDTOCHAR(rowid),

',' || to_char(pos_prod_rest_id) || ',' || 'PRES'

From pos_prod_rest_head

Where extract_req_ind = 'Y'

And effective_date = TO_DATE(:ps_tomorrow,'YYYYMMDD')

And ((NVL(:ps_restart_pos_config_type,' PRES') < 'PRES'

      OR(NVL(:ps_restart_pos_config_type,' PRES') = 'PRES'

      AND pos_prod_rest_id > :ps_restart_pos_config_id))

UNION ALL

Select coupon_id,

'COUP',

pos_config_status,

ROWIDTOCHAR(rowid),

',' || to_char(coupon_id) || ',' || 'COUP'

From pos_coupon_head

Where extract_req_ind = 'Y'
```

```

And effective_date = TO_DATE(:ps_tomorrow,'YYYYMMDD')

And ((NVL(:ps_restart_pos_config_type,' COUP') < 'COUP'

      OR(NVL(:ps_restart_pos_config_type,' COUP') = 'COUP'

      AND coupon_id > :ps_restart_pos_config_id))

UNION ALL

Select money_ord_id,

'MORD',

pos_config_status,

ROWIDTOCHAR(rowid),

',' || to_char(money_ord_id) || ',' || 'MORD'

From pos_money_ord_head

Where extreact_req_ind = 'Y'

And effective_date = TO_DATE(:ps_tomorrow,'YYYYMMDD')

And ((NVL(:ps_restart_pos_config_type,' MORD') < 'MORD'

      OR(NVL(:ps_restart_pos_config_type,' MORD') = 'MORD'

      AND money_ord_id > :ps_restart_pos_config_id))

ORDER BY 2,1;

```

Shared Modules

ITEMLOC_ATTRIB_SQL.ITEM_STORE_EXISTS – validates if an item/store relationship exists in the RMS.

Function Level Description

Main()

Perform standard main() function processing (logon, call init(), process(), and final() functions).

Init()

Perform standard Retek restart/recovery initialization (i.e. calling restart_file_init, initializing restart multithread variables, etc.). Select the vdate and place it in a variable called ps_vdate. Also figure vdate +1 and create a variable for that value and call it ps_prior_eff_date. This variable will be used in the driving cursor to select records whose effective day is the next day. Create a final and temporary (use the tmpname C function) flat file for output. Call the size_arrays() function to initialize the various arrays used within the program. Then, call the format_output() function to format the output strings for each record type in the flat file. Finally, write the FHEAD record to the output file.

Size_arrays()

Size the POS configuration, item fetch, and store arrays to the size of the restart_control.commit_max_ctr. If memory allocation fails, abort the program.

Format_output()

Format the output strings for each record type (see I/O Specification section for detail) written to the output file.

Process()

Within a loop, fetch the driving cursor into the POS configuration array. Invoke a for loop to process the array. Within the loop, call the retrieve_config() function to retrieve fields relative to the POS configuration type being processed. For example, if the POS config type is 'COUP' (coupons), go retrieve the necessary information from the coupon_head table.

Note: See the flat file to get an idea of what data needs to be selected off of each header table. Remember to select by rowid.

Next, if the POS configuration type is coupon or product restriction, call the free_linked_list_memory() to ensure that a clean linked list is started with for each configuration. Then for coupons and product restrictions call the process_criteria() and process_items() functions. These functions will populate an item array with all items that are associated with the current POS configuration. These item records will not be written to a flat file yet, but stored in a linked list to be downloaded later with the stores.

The `process_stores()` function will retrieve all stores tied to the POS configuration and place them in a store array. It will then loop through the store array, writing store records to the output file. In the case where the POS configuration type is coupons or product restrictions, it will also loop through items and write them to the flat file.

After processing the POS configuration and writing it to a flat file, call the `delete_records()` function to delete any records in 'D' status and to purge the `pos_config_items` table. Then, call the `restart_commit()` and `restart_file_write()` function will be called to commit the data and update the restart/recovery tables.

`Retrieve_config()`

Based on the current `pos_config_type` and `pos_cofnig_id` retrieve all information from the appropriate master table and write it to the flat file by calling the appropriate retrieve function based on the `pos_config_type`. If the `pos_config_type` is Payinout call the `retrieve_pyio_config()`. If the `pos_config_type` is Tender Type call the `retrieve_ttyp_config()`. If the `pos_config_type` is Buttons call the `retrieve_btn_config()`. If the `pos_config_type` is Supllier Payment Type call the `retrieve_spay_config()`. If the `pos_config_type` is Product Restriction call the `retrieve_pres_config()`. If the `pos_config_type` is Coupon call the `retrieve_coup_config()`. If the `pos_config_type` is Money Order call the `retrieve_mord_config()`.

`Retrieve_pyio_config()`

Retrieve all Payinout information from `pos_payinout_head` and write it to the flat file according to Retek standards.

`Retrieve_ttyp_config()`

Retrieve all Tender Type information from `pos_tender_type_head` and write it to the flat file according to Retek standards.

`Retrieve_btn_config()`

Retrieve all Button information from `pos_button_head` and `pos_button_detail` and write it to the flat file according to Retek standards.

`Retrieve_spay_config()`

Retrieve all Supplier Payment information from `pos_sup_pay_criteria` and write it to the flat file according to Retek standards.

`Retrieve_pres_config()`

Retrieve all Product Restriction information from `pos_prod_rest_head` and `pos_day_time_date` and write it to the flat file according to Retek standards.

Retrieve_coup_config()

Retrieve all Tender Type information from pos_coupon_head and write it to the flat file according to Retek standards.

Retrieve_mord_config()

Retrieve all Money Order information from pos_money_ord_head and write it to the flat file according to Retek standards.

Process_criteria

Fetch any changed merchandise criteria associated with the current POS configuration into an array. Since the number of records returned is variable and the information will not be processed until later (after the stores are selected), it will be loaded into a linked list. After the fetch, loop through the array and call the build_item_list() function to allocate memory and enter the item record into a linked list. Load the status field with the value of 'C' (changed).

The syntax of the select statement should be similar to the following:

```
select distinct dl.sku,
from desc_look dl,
      deps d,
      pos_merch_criteria p1
where dl.dept = d.dept
      and p1.dept = d.dept
      and p1.group_no = d.group_no
      and p1.pos_config_type = :ps_pos_config_type
      and p1.pos_config_id = :ps_pos_config_id
      and ( ( p1.dept is NULL)
            or ( dl.dept = p1.dept
                  and p1.classt is NULL)
            or ( dl.dept = p1.dept
                  and dl.class = p1.class
                  and p1.subclass is NULL)
```



```

or ( dl.dept      = p1.dept
    and p1.class   = dl.class
    and p1.subclass = dl.subclass
    and p1.sku      is NULL)
or ( dl.dept      = p1.dept
    and p1.class   = dl.class
    and p1.subclass = dl.subclass
    and p1.sku      = dl.sku
    and p1.exclude_ind = 'N')
and NOT EXISTS (select p2.pos_config_id
                from pos_merch_criteria p2
                where p2.sku = p1.sku
                   and p2.pos_config_id = :ps_pos_config_id
                   and p2.pos_config_type = :ps_pos_config_type
                   and p2.exclude_ind = 'Y'))

order by 1

```

The item linked list definition is as follows:

```
Typedef struct item_rec
```

```

{
    char as_item[LEN_ITEM];
    char as_status[LEN_STATUS];
    struct item_rec *ar_next_rec;
} item_linked_list;

```

Note: The as_store variable will only be populated from records added in the process_items() function.

```
Process_items
```

For each coupon and product restriction record, fetch any items that exist on the pos_config_items table. Since the process_criteria() function has potentially added items to the item linked list, a check must be made to determine if the item has already been added. It is also imperative the correct status is associated with the items. Below is a matrix indicating how different possible situations should be handled with the item array.

	Exist in Linked List (in 'C'hanged status)	Doesn't Exist in Linked List
Item in 'A'dd status	Do nothing.	Add item to linked list in 'A'dd status.
Item in 'D'elele status	Change status in linked list to 'D'	Add item to linked list in 'D'elele status.
Item in 'R'etail change status	Do nothing	Add item to linked list in 'R'etail status

The syntax of the select statement to retrieve items should be similar to the following:

Select item,

store,

status,

retail

From pos_config_items

Where pos_config_type = :ps_pos_config_type

And pos_config_id = :ps_pos_config_id;

Process_stores

If the POS configuration has changed, the merch criteria has changed, and/or, records exist on pos_config_items, select all stores from pos_store into the store array. If none of the conditions are met (i.e. only changes are stores being added to and deleted from the POS configuration) then select stores from pos_store where status is not null.

The store status will be set to 'A' when:

The POS configuration is new to the store (header.config_status = 'A').

The store is added to a POS configuration (pos_store.status = 'A').

The store status will be set to 'C' when:

The POS configuration has been modified (header.config_status = 'C').

The merchandise criteria has been modified (pos_merch_criteria.status = 'A','C', or 'D').

Additional items have been added/deleted/reclassified to POS configurations at the store (pos_config_items.status = 'A','D', 'R').

The store status will be set to 'D' when:

The POS configuration has been deleted (header.config_status = 'D').

The store is deleted from a POS configuration (pos_store.status = 'D').

The syntax of the select statement to retrieve stores should be similar to the following:

Select store,

status,

retail

From pos_store

Where pos_config_type = :ps_pos_config_type

And pos_config_id = :ps_pos_config_id

And status is NOT NULL;

In a loop, write the records from the store array to the flat file. If the POS configuration type is coupon or product restriction call the write_item_to_file() function to validate the item/store relationship and write items associated to the output file.

Write_item_to_file()

In a loop, call the ITEMLOC_ATTRIB_SQL.ITEM_STORE_EXISTS function to validate that the item/store exists in the RMS database. If the relationship exists, write the item to the flat file. If the relationship does not exist, skip the item and do not write it to the flat file.

Delete_records()

This function will delete records from the appropriate tables that are in delete status. It is imperative that tables are deleted in the correct order.

Delete pos_config_items for the pos_config_type/pos_config_id

Delete pos_store for the pos_config_type/pos_config_id and status is 'D'

Delete pos_merch_criteria for pos_config_type/pos_config_id and status is 'D'

Delete from all the header tables (9 tables) where status = 'D'

Note: In the case of Product Restrictions information from the pos_day_time_date table will need to be deleted from the system before the deletion from pos_prod_rest_head can occur. Delete all records from pos_day_time_date where pos_day_time_date.pos_prod_rest_id = pos_prod_rest_head.pos_prod_rest_id and pos_prod_rest_head.status = 'D'. Then continue with the deletion from pos_prod_rest_head.

Build_item_list()

This function will copy records from the items array to the items linked list. If the linked list is empty, define the current item record as the first record in the linked list and allocate memory for it. If the linked list is not empty, save the record address as the next record in the linked list, then move ahead to the next record in the linked list and allocate memory. After memory allocation has occurred, copy in the array record into the linked list. Finally, set the next record pointer to NULL.

Free_linked_list_memory()

Starting at the first record of the linked list, loop through the list and free memory until the last record is reached (current record pointer is NULL). When completed, set the first and current record pointers to NULL.

I/O Specification

Record type relationship

FHEAD

TCOUP

TSTOR

TITEM

TPROR

TSTOR

TITEM

TMORD

TSTOR

TTEND

TSTOR

TBTTN

TSTOR

TITEM

TPYIO

TSTOR

TSPPT

TSTOR

FTAIL

Record Type	Field	Datatype	Valid Values	Comments
FHEAD	Record Type	Char(5)	'FHEAD'	Record Identifier
	Line ID	Number(10)	0000000001	Sequential Line Identifier
	File Name	Char(4)	'POSC'	File Identifier
	File Date	Char(14)		Date the file was created in 'YYYYMMDDHHMMSS' format
TCOUP	Record Type	Char(5)	'TCOUP'	Record Identifier
	Line ID	Number(10)		Sequential Line Identifier
	Coupon ID	Number(6)		
	Coupon Desc	Char(40)		
	Currency Code	Char2(3)		
	Max Discount Amount	Number(20,4)		
	Amount	Number(20,4)		
	Percent Ind	Char(1)		
	Profit Center	Char(6)		
	Tax Class	Char(6)		
	Export Code	Char(6)		
	Effective Date	Char(14)		
	Expiration Date	Char(14)		
	Prompted Ind	Char(1)		

Record Type	Field	Datatype	Valid Values	Comments
	Display Ind	Char(1)		
	Status	Char(1)	'A','C','D'	Indicates if the POS configuration is new, has been changed, or being deleted.
TPROR	Record Type	Char(5)	'TPROR'	Record Identifier
	Line ID	Number(10)		Sequential Line Identifier
	POS Product Restriction ID	Number(6)		
	POS Product Restriction Desc	Char(40)		
	POS Product Restriction Type	Char(6)		
	Effective Date	Char(14)		
	Currency Code	Char(3)		
	Product Restriction Amount	Number(20,4)		
	Age Minimum	Number(2)		
	Date Restriction	Char(14)		
	Before Time Restriction	Char(6)		
	After Time Restriction	Char(6)		
	Day Restriction	Char(6)		
	Max Qty Amount	Number(12,4)		
	Tender Type Group	Char(6)		
	Status	Char(1)	'A','C','D'	Indicates if the POS configuration is new, has been changed, or being deleted.

Record Type	Field	Datatype	Valid Values	Comments
TMORD	Record Type	Char(5)	'TMORD'	Record Identifier
	Line ID	Number(10)		Sequential Line Identifier
	Money Order ID	Number(6)		
	Money Order Desc	Char(40)		
	Currency Code	Char(3)		
	Fee	Number (20,4)		
	Max Face Value	Number (20,4)		
	Max Sale Amount	Number (20,4)		
	Tax Class	Char (6)		
	Cash Only Ind	Char (1)		
	Serial Input Ind	Char(1)		
	Pack Size	Number (6)		
	Refund Fee Ind	Char (1)		
	Expiration Date	Char(14)		
	Effective Date	Char(14)		
	Status	Char(1)	'A','C','D'	Indicates if the POS configuration is new, has been changed, or being deleted.
TTEND	Record Type	Char(5)	'TTTYP'	Record Identifier
	Line ID	Number(10)		Sequential Line Identifier
	Tender Type ID	Char(6)		
	Tender Type Desc	Char (40)		
	Tender Type Group	Char (6)		
	Effective Date	Char(14)		

Record Type	Field	Datatype	Valid Values	Comments
	Currency Code	Char(3)		
	Preset Amount	Number(20,4)		
	Authorize Minimum	Number(20,4)		
	Open Drawer Ind	Char (1)		
	Exact Change Ind	Char (1)		
	Accumulate Cash Intake Ind	Char (1)		
	Next Dollar Ind	Char (1)		
	Deposit in Bank Ind	Char (1)		
	Deposit Override Ind	Char (1)		
	Automatic Deposit Ind	Char (1)		
	Pay In Deposit Ind	Char (1)		
	Ask For Invoice Ind	Char (1)		
	Imprint Ind	Char (1)		
	Imprint Tender Type	Char (20)		
	Show in Breakdown Ind	Char (1)		
	Display Ind	Char (1)		
	Discrepancy Display Type	Char (6)		
	Processor Type	Char (6)		
	Export Code	Char(6)		
	Profit Center	Char(6)		

Record Type	Field	Datatype	Valid Values	Comments
	Phone Authorization Type	Char(6)		
	Status	Char(1)	'A','C','D'	Indicates if the POS configuration is new, has been changed, or being deleted.
TBTTN	Record Type	Char(5)	'TBTTN'	Record Identifier
	Line ID	Number(10)		Sequential Line Identifier
	Button configuration ID	Number (6)		
	Button Config Desc	Char(40)		
	Button ID	Number (6)		
	Button Desc	Char (40)		
	Layer Number	Number (1)		
	Sequence Number	Number (4)		
	Button Type	Char(6)		Valid Values include: U = Multi Selection S = Single Selection I = Single Item M = Major Category N = Minor Category X = No Button P = Programmable
	Effective Date	Char(14)		
	Text Color	Char (6)		
	Background Color	Number (6)		
	Department	Char(4)		
	Text	Char (10)		
	Parent Button ID	Char(6)		

Record Type	Field	Datatype	Valid Values	Comments
	Status	Char(1)	'A','C','D'	Indicates if the POS configuration is new, has been changed, or being deleted.
TPYIO	Record Type	Char(5)	'TPYIO'	Record Identifier
	Line ID	Number(10)		Sequential Line Identifier
	Pay In Out ID	Number(6)		
	Pay In Out Desc	Char(40)		
	Pay In Out Type	Char(1)		
	Profit Center	Char(6)		
	Export Code	Char(6)		
	Tax Chart Code	Char(6)		
	Invoice Link	Char(1)		
	Display Ind	Char(1)		
	Effective Date	Char(14)		
	Status	Char(1)	'A','C','D'	Indicates if the POS configuration is new, has been changed, or being deleted.
TSPPT	Record Type	Char(5)	'TSPPT'	Record Identifier
	Line ID	Number(10)		Sequential Line Identifier
	Supplier	Number(10)		
	Pay Type	Char(6)		Valid Values include: CASH = Cash MP = Money Order INVC = Invoice
	Supplier Payment Type ID	Number(6)		
	Effective Date	Char(14)		

Record Type	Field	Datatype	Valid Values	Comments
	Status	Char(1)	'A','C','D'	Indicates if the POS configuration is new, has been changed, or being deleted.
TSTOR	Record Type	Char(5)	'TSTOR'	Record Identifier
	Line ID	Number(10)		Sequential Line Identifier
	Store	Number(4)		Store Identifier
	Status	Char(1)	'A' – Add 'D' – Delete	Indicates to the POS if the POS configuration must be added or deleted.
TITEM	Record Type	Char(5)	'TITEM'	Record Identifier
	Line ID	Number(10)		Sequential Line Identifier
	Item	Number(8)		
	Status	Char(1)	'A' – Add 'D' – Delete 'C' – Change	Indicates the item's status at the POS. Overlays of items as a result of a change to the merch criteria will have a 'C' status.

POS Download [posdnld]

Design Overview

The posdnld program is used to download pos_mods records created in the RMS to the store POS systems. This program has one output file which contains all records for all stores in a given run.

Scheduling Constraints

Processing Cycle: PHASE 4 (daily)

Scheduling Diagram: This program is run towards the end of the batch run when all pos_mods records have been created for the transaction day.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

Restart/recovery for this program is set up at the store/upc or sku level. Threading is done by store using the v_restart_store view to thread properly.

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

Program Flow



Shared Modules

N/A

Function Level Description

Init

This function initializes restart/recovery for this program. It also opens the output file, retrieves system variables and calls a function to size the arrays used in this program.

Process

This function drives the processing of the program. The driving cursor is fetched here which retrieves all the records from pos_mods where the pos_mods.store value is greater than zero. Once the records are fetched, the write_rec() function is called to perform processing on them. Restart/Recovery and committing of records is also performed here.

Write_rec

This function will prepare records for insert into the output file. This program used the Retek standard file format FHEAD, FDETL, FTAIL.

Final

This function will finish restart/recovery logic, close the output file and delete the temporary output file used while the program processes.

Init_format_strs

This function formats the strings for the FHEAD, FDETL, and FTAIL records in the output file.

Init_arrays

This function initializes the size of the array used for the driving cursor fetch the size of the restart max counter on restart_control.

Resize_arrays

This function increases the memory for the driving cursor array by the size of the restart max counter on restart_control.

Get_upc_type

This function query the upc_ean table to get the var_type given the passed in upc and upc_supp.

I/O Specification**Output file**

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)	Sequential number Created by program.	ID of current line being created for output file.
	File Type Definition	Char(4)	POSD	Identifies file as 'POS Download'
	File Create Date	Char(14)	create date	current date, formatted to 'YYYYMMDDH H24MISS'.
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	Sequential number Created by program.	ID of current line being created for output file.
	Location Number	Number(4)	store identifier	Store identifier from the store table in Retek

Record Name	Field Name	Field Type	Default Value	Description
	Update Type	Char(1)	update type	Code used for client specific POS system.
	Start_Date	Char(14)	start date	date for the change to take effect at the POS, formatted to 'YYYYMMDDHH24MISS'.
	Upc	Number(13)	upc identifier	the id number of a UPC. From the Retek upc_ean table.
	Upc Supplement	Number(5)	supplemental identifier	used to further specify the id of an UPC item. From the Retek upc_ean table.
	Upc var type	Char(1)	Variable upc type indicator	Identifies what type of variable UPC is being sent. Valid values are 'W'eight, 'P'rice, and NULL.
	Tran Type	Number(2)	transaction type	the transaction type for the record from the Retek pos_mods table.
	SKU	Number(8)	SKU identifier	the id number of a SKU from the Retek desc_look table.
	SKU Description	Char(40)	SKU description	the description of the SKU from the Retek desc_look table.
	Dept	Number(4)	dept id	the id of the dept for the item from the Retek win_skus or rag_style table.

Record Name	Field Name	Field Type	Default Value	Description
	Class	Number(4)	class id	the id of the class for the item from the Retek win_skus or rag_style table.
	Subclass	Number(4)	subclass id	the id of the subclass for the item from the Retek win_skus or rag_style table.
	New Price	Number(20)	new price	the new price to be taken at the POS. This value is from the Retek pos_mods table.
	Multi Units	Number(12)	multi units	Number of multi units
	Multi Units Retail	Number(20)	multi units retail	unit retail for the multi units
	Status	Char(1)	status	Populates if tran_type for the item is 1(new item added) or 25 (change item status) or 26 (change taxable indicator).
	Taxable Indicator	Char(1)	taxable ind	Populates if tran_type for the item is 1(new item added) or 25 (change item status) or 26 (change taxable indicator).
	Promotion Number	Number(4)	promotion number	Promotion number for SKU. This value is from the Retek system.
	Mix Match Number	Number(4)	mix match number	mix match number for SKU. This value is from the Retek ssystem.

Record Name	Field Name	Field Type	Default Value	Description
	Mix Match Type	Char(1)	mix match type	mix match type (Buy or Get) for SKU. This value is from the Retek system.
	Threshold Number	Number(4)	threshold number	Threshold number for SKU. This value is from the Retek system.
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	Sequential number Created by program.	ID of current line being created for output file.
	File Record Counter	Number(10)		Number of records/transactions processed in current file (only records between head & tail)

Technical Issues

N/A

[posgpdlld.pc]

Design Overview

Occasionally, retailers need to perform department level sales for merchandise. To accommodate this, a new download program (posgpdlld.pc) is created to download group number/department information to individual Point-of-Sale (POS) systems.

Assumption: The dataset that defines group numbers and departments is small. With this assumption, a full extract of RMS group number/department combinations occurs for every run of the batch program.

Scheduling Constraints

Processing Cycle:	ad hoc
Scheduling Diagram:	Run after reclsdly.pc
Pre-Processing:	reclsdly.pc
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

Logical Unit of Work

Group_no/dept

Driving Cursor

```
SELECT group_no,
       dept,
       ','||to_char(group_no)||';'||to_char(dept)
FROM deps
WHERE (group_no > NVL(:ps_restart_group_no, -999)
      OR (group_no = :ps_restart_group_no
          AND dept > :ps_restart_dept))
ORDER BY group_no, dept;
```

Shared Modules

N/A

Function Level Description

Global structure of arrays for data processing:

```
struct fetch_rec  
{  
    char (*as_group_no)[NULL_GROUP_NO];  
    char (*as_dept)[NULL_DEPT];  
    char (*as_restart_string)[NULL_RESTART]  
};
```

Called from init():

int get_date()

Fetches the vdate (business date) from the period table and places it into a program global variable.

int size_array(struct fetch_recs *ioa_data_buffer)

Sizes the arrays used in database FETCHES by using the calloc function.

void format_strings()

Creates format strings used in file outputs.

int init()

Performs standard Retek restart/recovery initialization. All file handling occurs here, such as declaration of a temp file, and opening of files. The FHEAD line is also written to the file, in the case of a fresh start.

int process()

The driving cursor is used to fetch records from the database. The group_no and dept are written to the temp file using the FDETL format string defined previously. Once all data has been fetched an FTAIL line is written to the file.

int final()

Performs closing operations. A call to restart_close() is made, and the temp file is removed from the system.

I/O Specification

Record Type	Field	Datatype	Valid Values	Description
FHEAD	Record Type	Char(5)	'FHEAD'	Record Identifier
	Line ID	Number(10)	0000000001	Line Sequence Identifier
	File Name	Char(4)	'GRPD'	File Identifier
	File Date	Char(14)		Date the file was created in 'YYYYMMDDHHMMSS' format
FDETL	Record Type	Char(5)	'FDETL'	Record Identifier
	Line ID	Number(10)		Line Sequence Identifier
	Group no	Numeric ID (4) (left-justified character field)		Valid RMS Group Number.
	Dept	Numeric ID (4) (left-justified character field)		Valid RMS Department for the Group Number.
FTAIL	Record Type	Char(5)	'FTAIL'	Record Identifier
	Line ID	Number(10)		Line Sequence Identifier
	Line Total	Number(10)		Total number of FDETL lines in the file.

POS Upload [posupld]

Design Overview

The purpose of this batch module is to process sales and return details from an external point of sale system. The sales/return transactions will be validated against Retek sku/store relations to ensure the sale is valid, but this validation process can be eliminated if the sales being passed in have already been screened by sales auditing. The following common functions will be performed on each sales/return record read from the input file:

- read sales/return transaction record
- validate item sale
- check if VAT maintenance is required, if so determine the VAT amount for the sale
- write all financial transactions for the sale and any relevant markdowns to the stock ledger.
- post SKU/location/week sales to the relevant sales history tables
- if a late posting occurs in a previous week (i.e. not in the current week), if the item for which the late posting occurred is forecastable, the last_sales_export_date on the item store tables has to be updated to the end of week date previous to the week of the late posting. This will result in the sales download interface programs extracting the week(s) for which the late transactions were posted to maintain accurate sales information in the external forecasting system.

Scheduling Constraints

Processing Cycle: PHASE 2 (daily)

Scheduling Diagram: This program will likely be run at the beginning of the batch run during the POS polling cycle. It can be scheduled to run multiple times throughout the day, as POS data becomes available.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

The logical unit of work for the sales/returns upload module will be a valid SKU sales transaction at a given store location. The location type will be inferred as a store type and the item can be passed as a SKU or UPC type. The logical unit of work will be defined as a number of these transaction records. The `commit_max_ctr` field on the `restart_control` table will determine the number of transactions that equal a logical unit of work.

The file records will be read in groups of numbers equal to the `commit_max_ctr`. After all records in a given read are processed (or rejected), the restart commit logic and restart file writing logic will be called, and then the next group of file records will be read and processed. The commit logic will save the current file pointer position in the input file and any application image information (e.g. record and reject counters) and commit all database transactions. The file writing logic will append the temporary holding files to the final output files.

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

Error handling will recognize three levels of record processing: process success, non-fatal errors, and fatal errors. Item level validation will occur on all fields before table processes are initiated. If all field-level validations return successfully, inserts and updates will be allowed. If a non-fatal error is produced, the remaining fields will be validated, but the record will be rejected and written to the reject file. If a fatal error is returned, then file processing will end immediately. A restart will be initiated from the file pointer position saved in the `restart_bookmark` string at the time of the last commit point that was reached during file processing.

Program Flow

N/A

Shared Modules

`validate_all_numeric`: intrface library function.

`validate_all_numeric_signed`: intrface library function.

`valid_date`: intrface library function.

`ORDER_ATTRIB_SQL.DELIVERY_MONTH`: called from `consignment_data()`, returns order delivery month into the `:invoices` variable.

`VAT_SQL.GET_VAT_RATE`: called from `pack_check()`, returns the composite vat rate for a packitem.

CURRENCY_SQL.CONVERT: returns the converted monetary amount from Currency to currency.

NEW_STAPLE_LOC: called from win_check(), creates a new staple SKU if one doesn't already exist for the SKU/location passed in.

NEW_FASHION_LOC: called from rag_check(), creates a new fashion SKU if one doesn't already exist for the SKU/location passed in.

NEW_PACK_LOC: called from pack_check(), creates a new packitem if one doesn't already exist for the packitem/location passed in.

UPDATE_SNAPSHOT_SQL.EXECUTE: called from update_snapshot(), updates the stake_sku_loc and edi_daily_sales tables for late transactions. If the item is a return, edi_daily_sales will not be updated.

NEXT_ORDER_NO: called from consignment_data(), returns a randomly generated order number.

STKLDGR_SQL.TRAN_DATA_INSERT: called from consignment_data(), performs tran_data inserts (tran_type 20) for a consignment transaction.

Function Level Description

Declarations:

declare input structures: file header (only date and type) & detail (all fields)

init()

initialize restart recovery

open input file (posupld)

- file should be specified as input parameter to program

fetch system variables

Retrieve all valid promotion types

declare final output filename (used in restart_write_file logic)

open reject file (as a temporary file for restart)

- file should be specified as input parameter to program

call restart_file_init logic

assign application image array variables- line counter (g_l_rec_cnt), reject counter (g_l_rej_cnt), store, transaction_date

```
if fresh start (l_file_start = 0)

read file header record (get_record)

if (record type <> 'FHEAD') Fatal Error

validate file type = 'POSU'

else fseek to l_file_start location

validate location and date are valid

file_process()
```

This function will perform the primary processing for transaction records retrieved from the input file. It will first perform validation on the THEAD record that was fetched. If the transaction was found to be invalid, a record will be written to the reject file, a non-fatal error will be returned, and the next transaction will be fetched.

Next, the unit retail from price_hist will be fetched by calling the get_unit_retail() function. The retail retrieved from this function will be compared with the actual retail sent in from the input file to determine any discrepancies in sale amounts.

Fetch all of the TDETL records that exist for the transaction currently being processed until a TTAIL record is encountered. Perform validation on the transaction detail records. If a detail record is found to be invalid, the entire transaction will be written to the reject file, a non-fatal error will be returned, and the next record will be fetched. If a valid promotion type (code for mix & match, threshold promotions, etc.) was included in the detail record and it is not an employee disc record, write a record to the daily_sales_discount table. If it is an employee discount record write an employee discount record to tran_data. Finally, accumulate the discount amounts for all transaction detail records for the current transaction, unless the record was an employee discount.

Call the sku_process() function to perform item specific processing. Once all records have been processed, call posting_and_restart to commit the final records processed since the last commit and exit the function.

```
sku_process()
```

Set the item sales type for the current transaction. Valid sales types are 'R'egular sales, 'C'learance sales, and 'P'romotional sales. These will be used when populating the sales types for the item-location history tables. If an item is both on promotion and clearance, the transaction will be written as a clearance transaction.

If the system's VAT indicator is turned to on, VAT processing will be performed. The function `vat_calc()` will retrieve the vat rate and vat code for the current item-location. The total sales including and excluding VAT will be calculated for use in writing transaction data records. If any VAT errors occur, the entire transaction will be written to the reject file, a non-fatal error will be returned, and the next record will be fetched. A record will be written to `vat_history` for the item, location, transaction date.

Calculate the item sales totals (i.e. total retail sold, total quantity sold, total cost sold, etc.). If VAT is turned on in the system, calculate exclusive and inclusive VAT sales totals.

Calculate any promotional markdowns that may exist by calling the `calc_prom_totals()` function. The markdown information calculated here will be used when writing `tran_data` (`tran_type` 15) records for promotional markdowns.

Calculate the over/under amount the item was sold at compared to its `price_hist` record. Since we do not create `price_hist` records of type 9 (promotional retail change) when the `system_options.multi_prom_ind = 'Y'`, we do not know what the promotional retail for this item is. Therefore, we will take the total sales reported from the header record plus the total of sales discounts reported in the TDETL records, divided by the total sales quantity for the item to calculate its unit retail. If the `system_options.multi_prom_ind = 'N'`, we can do a comparison of the `price_hist` record and the unit retail (total retail / total sales) inputted from the POS file. Any difference using either method will write to the `daily_sales_discount` table with a promotion type of 'in store' and `tran_data` (`tran_type` 15). If the transaction is a return, no `daily_sales_discount` record will be written, and `tran_data` records will be written as opposite of what they were sold as (i.e. if the sale was written as a markup, which would be written as a negative retail with a `tran_data` 15, the return would be written as a 15 with a positive retail).

If the item is a packitem and the transaction is a Sale, the `process_pack()` function will update the `last_sale` field on the `packstore` table to the transaction date and the `packstore_hist` table will be updated with the transaction information.

If the item currently being processed is a packitem, calculate the retail markdown the item takes for being included in the pack and write a transaction data record as a promotional markdown. This markdown is calculated by comparing the retail contribution of the packitem's component SKU to the packitem to the component SKU's regular retail found on the `price_hist` table. The retail contribution for a component SKU is calculated by taking the component SKU's unit retail from `price_hist`, divided by the total retail of all component SKUs in the packitem, and multiplying the packitem's unit retail. So if the retail contribution of a component SKU within packitem A is \$10, and the same component SKU's `price_hist` record has a retail of \$14, and there is only one packitem sold, and this component SKU has a quantity of one, a `tran_data`

Record (`tran_type` 15) will be written for \$4 (assume no vat is used).

Write transaction data records for sales and returns. If the transaction is a sale, write a `tran_data` record with a transaction code of 1 with the total sales. If the system VAT indicator is on and the `system_options.stkldgr_vat_incl_retl_ind` is on, write a `tran_data` record with a transaction code of 2 for VAT exclusive sales. If the transaction is a return, write a `tran_data` record (`tran_type` 1) with negative quantities and retails for the amount of the return. If the system VAT indicator is on and the `system_options.stkldgr_vat_incl_retl_ind` is on, write a `tran_data` record (`tran_type` 2) and negative quantities and retails for the VAT exclusive return. Also, write a `tran_data` record with a transaction code of 4 for the total return. Any `tran_data` record that is written should be either VAT exclusive or VAT inclusive, depending on the `system_options.stkldgr_vat_incl_retl_ind`. If it is set to 'Y', all `tran_data` retails should be VAT inclusive. If it is set to 'N', all `tran_data` retails should be VAT exclusive. When writing `tran_data` records for packitems, always break them down to the packitem level, writing the retail as the packitem multiplied by the component SKU's price ratio. The packitem itself should never be inserted into the `tran_data` table.

If the transaction is late (transaction date is before the current date), call `update_snapshot()` to update the `stake_sku_loc` and `edi_daily_sales` tables. If the transaction is current, update the `edi_daily_sales` table only (`stake_sku_loc` will be updated in a batch program later down the stream). The `edi_daily_sales` table should only be updated if the items supplier edi sales report frequency = 'D'.

If VAT is turned on in the system, write a record to the `vat_history` table to record the vat amount applied to the transaction. The VAT amount is calculated by taking the sales including VAT minus the sales excluding VAT.

Update the sales history tables for non-consignment items that are Sale transactions. Do not update for returns. Also, update stock count on the item-location table for Sales and Returns unless the item is on consignment.

If an `off_retail` amount was identified for the item/location, call the `write_off_retail_markdowns()` function to write `tran_data` records (`tran_type` 15) to record the difference. If the `system_options.multi_prom_ind` = 'N' and the item is on promotion, or if the `system_options.multi_prom_ind` = 'Y' and the TDETL total discount amount is greater than zero, write a promotional markdown. Note: this will also record a `tran_data` record (`tran_type` 15) for a TDETL record that has a promotional transaction type with no promotion number in order to record the markdown.

If an employee discount TDETL record has been encountered, a `tran_data` record with `tran_code` 60 will be written.

If the item is a wastage item, a `tran_data` record with `tran_code` 13 will be written. This record is used to balance the stock ledger, it accounts for the amount of the sku that was wasted in processing.

`process_detail_error()`

This function writes a record to the `load_err` table for every non-fatal error that occurs.

`set_counters()`

Depending on the action passed into this function, it will either set a savepoint and store the values of counters or rollback a savepoint and reset the values of certain counters back to where they were originally set. This function is called when a non-fatal error occurs in the `sku_process()` function to rollback and changes that may have been made.

`calc_item_totals()`

This function will set total retail and discount values including and excluding VAT, depending upon the `store.vat_include_ind`, `system_options.vat_ind`, `system_options.multi_prom_ind`, and the `system_options.stkldgr_vat_incl_retl_ind`.

`calc_prom_totals()`

This function will set promotional markdown values including and excluding VAT, depending upon the `system_options.multi_prom_ind` and the `system_options.stkldgr_vat_incl_retl_ind`. If the `multi_prom_ind` is on, the promotional markdown is the sum of the TDETL discount amounts. If the `multi_prom_ind` is off, the promotional markdown is the difference between the `price_hist` record with a `tran_code` of 0,4,8,11 and the `price_hist` record with a `tran_code` of 9 multiplied by the total sales quantity. Also, the `tran_data` old and new retail fields are only written if the `multi_prom_ind` is off.

`process_sales_and_returns()`

If the item is on consignment and not a packitem, the `consignment_data()` function will be called to perform consignment processing. The function `write_tran` will be called to write a `tran_data` record with a `tran_type` 1 (always written), a `tran_type` 2 (if the `system_options.stkldgr_vat_incl_retl_ind = Y`), and a `tran_type` 4 (if the transaction was a return). If the transaction is a return, any `tran_data` records with `tran_types` of 1 and 2 will be written with negative retails. Also the `update_price_hist()` function will be called to update the most recent `price_hist` record.

`posting_and_restart()`

Post all array records to their respective tables and call `restart_file_commit` to perform a commit the records to the database and `restart_file_write` to append temporary files to output files.

`validate_FHEAD()`

Do standard string validations on input fields. This includes nullpadding fields, left shifting fields, checking that numeric fields are all numeric, and validating the date field. If any errors arise out of these validation checks, return non-fatal error then set non-fatal error flag to true. This function will also validate the store location exists.

If the sales audit indicator is on currency and vat information will be provided in the file that has already been validated.

validate_THEAD()

Do standard string validations on input fields. This includes nullpadding fields, left shifting fields, checking that numeric fields are all numeric, placing decimal in all quantity and value fields, and validating the date field. If any errors arise out of these validation checks, return non-fatal error then set non-fatal error flag to true. This function will also validate the UPC exists. If the upc_sup field is blank, set it to 00000.

If an UPC is passed in from the input file, retrieve the SKU for the UPC and UPC supplement. Once the item is a SKU, retrieve the system indicator, department, class, subclass, waste_type, waste_pct. Once this information is retrieved, check that the item/location relationship exists for the appropriate item type.

If the sale audit indicator is 'Y' on system_options, the item will be a SKU and the dept, class, subclass, and system_ind will be included in the file.

If an item is a wastage item set the wastage qty. The qty sent in the file shows the weight of the item sold. The wastage qty is the qty that was processed to come up with the qty sold. So if .99 of an item was sold, and item wastage percent is 10. The wastage qty is $.99 / (1-.10) = 1.1$. The wastage qty will be used through out the program except when writing tran_data records(see write_wastage_markdown) and daily_sales_discount records which will uses the processed qty from the file.

Get_upc_wt()

Assumption: Given the structure of the var_upc_ean table, it is only possible to have 100 different records on this table. This function uses this assumption when allocating memory to hold all the records on the var_upc_ean table. The logic will be required enhancements to accommodate a larger number of different variable weight upc types if desired.

Assumption: Non-variable weight UPC's can't start with a prefix that exists on the var_upc_ean table.

Assumption: All values passed in through variable weight UPC's will be considered to be in the standard unit of measure.

This function will extract an amount from a variable weight UPC. The first time this function is called it will build an array containing the var_upc_ean table. It will then proceed to search that array for a record that has a prefix matching the first two values in the UPC from the file. If it finds a matching record, it will use the information in the var_upc_ean array to strip out the weight from the UPC. It will then replace the characters of the UPC that held the weight with zeros so the sku can be retrieved from the UPC.

Example:

The UPC 2712345000000 is stored in RMS as a variable weight UPC. The prefix of the UPC is 27. The item identifier is 12345. The next five digits are the variable weight portion of the UPC. And finally the last character is a check digit for data transmission use.

The UPC is sent down to the stores and thus the scales with a variable weight indicator. This indicator lets the scales know to insert the weight sold, of the item, into the variable weight portion of the UPC upon a sale (with 3 implied decimal places). Let's say 7.7 pounds of our UPC were sold. The UPC would be uploaded from the POS file as 2712345077001. This function would strip out the 7.7 and assign it to the total qty sold variable. It would then replace the variable weight portion and the check digit of the UPC with zeros: 2712345000000. The UPC with the weight and check digit stripped out then will be used with the upc_ean table to get the sku.

validate_TDETL

Assumption: Currently posupld.pc can not interface with sales audit. This is due to the variable weight UPC logic. Sales audit currently doesn't recognize variable weight UPC's, and thus can not process them. The code designed to interface with Sales Audit is commented out. It should be uncommented when sales audit is updated to deal with variable weight UPC's.

Assumption: Variable weight UPC's need to be sent in at the transaction level, not the rolled up level that posupld usually receives. This is due to the fact that TDETL lines need to source their qty's from the UPC and not from the specified qty input fields.

This function will perform validation on the TDETL records passed into the program. The standard string validation on these fields includes nullpadding fields, left shifting fields, checking that numeric fields are all numeric, placing decimal in all quantity and value fields, and validating the date field. If any errors arise out of these validation checks, return non-fatal error then set non-fatal error flag to true.

If a promotional transaction type is passed in, verify it is valid. If a promotional transaction type is passed in, but it is not valid, return non-fatal error then set non-fatal error flag to true. If a promotion number is passed in, validate it by checking the promhead table and set the promotional indicator to True.

If the item is a wastage item set the tdetl wastage qty. This is done the same way as setting the THEAD wastage qty.

New_staple_loc

This function creates a new store sku relationship for staple items.

Win_store_cursors

This function checks the win_store for the sku / store combination. It is called by the win_check function.

win_check

This function verifies the staple item/location relationship exists. It is only called when the item being processed is a staple item. If the item/location relationship does not exist, it is created and a record is written to the Invalid item/location output file.

New_fashion_loc

This function creates a new store sku relationship for fashion items. It is called by rag_check.

rag_store_cursors

This function checks the win_store for the sku / store combination. It is called by the win_check function.

rag_check

This function verifies the fashion item/location relationship exists. It is only called when the item being processed is a fashion item. If the item/location relationship does not exist, it is created and a record is written to the Invalid item/location output file.

New_pack_loc

This function creates a new store sku relationship for fashion items. It is called by pack_check.

pack_check

This function verifies the pack item/location relationship exists and retrieves the component SKUs for the packitem. It is only called when the item being processed is a packitem. The component SKU, system indicator, department, class, subclass, cost, retail, price_hist retail, and component SKU quantity are fetched. If the packitem/location relationship does not exist, it is created for the Packitem and all of its components and a record is written to the Invalid item/location output file for the packitem.

The component SKUs price ratios are also calculated. This indicates the retail contribution the component SKU gives towards the unit retail of the packitem. This ratio is calculated by taking the price_hist unit retail of the component divided by the total price_hist retail of all the component SKUs for the packitem. Below is an example of how this ratio is calculated:

	Unit Retail	Qty	Retail	Calculation	Ratio
packitem A	\$60				
SKU 1	\$15	2	\$30	$(\$30/\$90) * \$60$	.3333
SKU 2	\$10	6	\$60	$(\$60/\$90) * \$60$	.6667

get_unit_retail

This function retrieves the unit retail from price_hist for the item/location being processed. If the item being processed is not a component SKU to a packitem that is currently being processed and the transaction is a sale, call get_all_price. If that is not found, call get_base_price. If the price_hist record fetched is a 9 (promotional retail change), the item being processed is a component SKU to a packitem currently being processed, or the transaction is a return, call get_reg_price. (these are base retail changes). If a tran_code of 8 is returned, the item is on clearance.

Get_all_price

This function will get the retail from price_hist for tran_types in (0,4,8,9,11). It is called by get_unit_retail.

Get_reg_price

This function will get the retail from price_hist for tran_types in (0,4,8,11). It is called by get_unit_retail.

Get_base_price

This function will get the retail from price_hist for tran_types of 0. It is called by get_unit_retail.

process_packitems

This function performs processing for the component SKUs of the packitems. This would include updates/inserts into stake_sku_loc, edi_daily_sales, win_store, rag_skus_st, win_store_hist, rag_skus_st_hist, vat_history_data, and tran_data. All of these tables do not write records at the packitem level, but at the component SKU level. When figuring retails to write to these tables, the component SKUs price ratio should always be applied against the packitems retail to come up with the correct retail for each component SKU. If an employee discount TDETL record has been encountered, an tran_data record with tran_code 60 will be written for each component sku.

`process_daily_sales_discount()`

This function will insert/update a record to `daily_sales_discount` for each TDETL record that has a promotional transaction type except employee discounts. Employee discount records are not written to `daily_sales_discount`, they are put on `tran_data` with a `tran_code` of 60. When employee discount records are encountered, values are set for the `tran_data` insert and the discount amount is added to the total sales value. This is done so employee discounts do figure into the promotional and in store calculations. When the `multi_prom_ind` is on all promotion types except employee discount will be ignored.

`write_in_store()`

This function will handle record sent in as 'is store' discounts amounts. It will call `check_daily_exist` and `daily_sales_insert_update`.

`Remove_stklgdr_vat()`

This function will remove vat from 3 fields after the `daily_sales_discount` processing is complete. The variables `od_off_retail_amt`, `od_new_retail`, and `od_old_retail` are stripped of vat by calling `vat_convert` if the stock ledger does not contain vat.

`Write_off_retail()`

This function will calculate discrepancies between the amount sold for an item, and the amount it should have sold for (`price_hist` record). If these amounts are not in balance, a record is written to the `daily_sales_discount` table with a `prom_type` of 'in store' for reporting.

`Daily_sales_exist()`

This function will check the `daily_sales_discount` for the existence of a record matching the input parameters

`Daily_sales_insert_update()`

This function is called by `write_off_retail`, `write_in_store`, and `process_daily_sales_discount`. It performs the actual insert or fills a update array for the `daily_sales_discount` table.

`write_off_retail_markdown()`

The `write_tran_data()` function will be called to write the off_retail markdown unless the item is on consignment or the off_retail amount is zero.

`write_promotional_markdown()`

The `write_tran_data()` function will be called to write the promotional markdown unless the item `multi_prom_ind` is off and the transaction is a return, the item is on consignment, or the promotional markdown amount is zero. The `tran_data` new and old retails are only written if the `multi_prom_ind` is off.

`Write_wastage_markdown()`

This function will call to the `write_tran_data()` function if the item is a wastage item. A wastage item is an item that loses some of its weight (value) in processing. For example, a 10 pound chicken is broiled and loses 10% of its weight. The item is sold at .9 pounds, but in reality selling that .9 pounds of chicken removes 1 pound of chicken from the inventory. This function writes a `tran_code` 13 `tran_data` record to account for the amount of the chicken that was lost due to wastage in processing.

`vat_convert()`

This function will either add or remove vat from a retail value.

`process_win()`

Update the stock on hand on the `win_store` table for Sales and Returns unless the item is on consignment. Also, update the `win_store_hist` table for Sale transactions. Do not update for returns.

`process_rag()`

Update the stock on hand on the `rag_skus_st` table for Sales and Returns unless the item is on consignment. Also, update the `rag_skus_st_hist` table for Sale transactions. Do not update for returns.

`process_pack()`

Update the stock on hand on the `packstore` table for Sales and Returns. Also, update the `rag_skus_st_hist` table for Sale transactions. Do not update for returns.

`write_tran_data()`

Writes a record to the `tran_data` insert array.

`Write_edi_daily_sales()`

Writes a record to `edi_daily_sales`.

`update_snapshot()`

Calls the `UPDATE_SNAPSHOT_SQL.EXECUTE` function to update the `stake_sku_loc` and `edi_daily_sales` tables for late transactions.

`write_vat_err_message()`

This function will create and write to the VAT output file when an item does not have VAT information setup when it is expected.

`vat_history_data()`

Writes a record to the `vat_history` table.

`consignment_data()`

This function will perform processing for consignment items. Consignment items are such when the `item_supplier` table has a consignment rate applied to it. Consignment is when a retailer will allow a third party to operate under its umbrella and be paid for what it sells. An example of consignment may be a mass-merchant who consigns the magazine section of their store to a magazine vendor. The magazine vendor would have control over keeping the product stocked within the store. When a magazine is sold, the retailer would get paid for the magazine, then the retailer would essentially buy the magazine from the vendor. The consignment cost paid by the retailer to the vendor is the VAT-inclusive retail multiplied by the consignment rate divided by 100. So if the VAT-inclusive retail price of a magazine was \$10 and the consignment rate was 50, the consignment cost would be \$5.

Also a completed order to the vendor should be found/created for the supplier with an `orig_ind = 4` (consignment). Consignment type invoices will be created for all PO's created for consignments

Also a `tran_data` record (`tran_type 20`) will be written to record the consignment transaction to the stock ledger. The retails should be VAT inclusive or exclusive, depending on the `system_options.stkldgr_vat_incl_retl_ind`.

This function uses support functions: `check_order()`, `order_head()`, `invc_data()`, to handle the order creation-update and the invoice creation-update.

`get_prom_type_info()`

This function will retrieve all valid promotional transaction types from the `code_detail` table. Valid promotional transaction types are those where the `code_type = 'PRMT'`.

`fill_packitem_array()`

This function will retrieve the component SKUs for a packitem with the appropriate item level information into an array.

`write_sku_store_report()`

This function will create and write to the Invalid item/location output file when an item does not exist at a location it was sold/returned at.

ON Fatal Error

- Exit Function with -1 return code

ON Non-Fatal Error

- write out rejected record to the reject file using write_to_rej_file function, pass pointer to detail record structure, number of bytes in structure, and reject file pointer

Input File

The input file should be accepted as a runtime parameter at the command line. All number fields with the number(x,4) format assume 4 implied decimal included in the total length of 'x'.

When the system_options field sa_ind is 'Y' the following FHEAD fields will be populated and already validated: Vat include indicator, Vat region, Currency code, and Currency retail decimals. When the sa_ind is 'N' these values will not be used and retrieved from the system.

When the system_options field sa_ind is 'Y' the following FHEAD fields will be populated and already validated: System_ind, Dept, Class, and Subclass. When the sa_ind is 'N' these values will not be used and retrieved from the system. When the sa_ind is on, all items are assumed to be SKUs.

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	Char(10)	specified by external system	ID of current line being processed by input file.
	File Type Definition	Char(4)	POSU	Identifies file as 'POS Upload'
	File Create Date	Char(14)	create date	date file was written by external system
	Location Number	Number(4)	specified by external system	Store or warehouse identifier

Record Name	Field Name	Field Type	Default Value	Description
	Vat include indicator	Char(1)		Determines whether or not the store stores values including vat. Not required but populated by Retek sales audit
	Vat region	Number(4)		Vat region the given location is in. Not required but populated by Retek sales audit
	Currency code	Char(3)		Currency of the given location. Not required but populated by Retek sales audit
	Currency retail decimals	Number(1)		Number of decimals supported by given currency for retails. Not required but populated by Retek sales audit
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies transaction record type
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	Transaction Date	Char(14)	transaction date	date sale/return transaction was processed at the POS
	Item Type	Char(3)	UPC SKU	item type will be represented as an UPC, an SKU
	Item Value	Char(13)	item identifier	the id number of a SKU or UPC

Record Name	Field Name	Field Type	Default Value	Description
	Supplement	Char(5)	supplemental identifier	used to further specify the id of an UPC item, or the pre-pack id reference
	System_ind	Char(1)	'S'- staple sku 'F'- fashion sku 'P'- pack item	The type of item sold or returned. Not required but populated by Retek sales audit
	Dept	Number(4)	Item's dept	Dept of item sold or returned. Not required but populated by Retek sales audit
	Class	Number(4)	Item's class	Class of item sold or returned. Not required but populated by Retek sales audit
	Subclass	Number(4)	Item's subclass	Subclass of item sold or returned. Not required but populated by Retek sales audit
	Wastage Type	Char(6)	Item's wastage type	Wastage type of item sold or returned. Not required but populated by Retek sales audit
	Wastage Percent	Number(12)	Item's wastage percent	Wastage percent of item sold or returned. Not required but populated by Retek sales audit
	Transaction Type	Char(1)	'S' – sales 'R' - return	Transaction type code to specify whether transaction is a sale or a return

Record Name	Field Name	Field Type	Default Value	Description
	Total Sales Quantity	Number(12)		Number of units sold at a particular location with 4 implied decimal places.
	Sales Sign	Char(1)	'P' - positive 'N' - negative	Determines if the Total Sales Quantity and Total Sales Value are positive or negative.
	Total Sales Value	Number(20)		Sales value, net sales value of goods sold/returned with 4 implied decimal places.
	Last Modified Date	Char(14)		For VBO future use
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies transaction record type
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	Promotional Tran Type	Char(6)	promotion type – valid values see code_detail table.	code for promotional type from code_detail, code_type = 'PRMT'
	Promotion Number	Number(4)	promotion number	promotion number from the RMS
	Sales Quantity	Number(12)		number of units sold in this prom type with 4 implied decimal places.

Record Name	Field Name	Field Type	Default Value	Description
	Sales Value	Number(20)		value of units sold in this prom type with 4 implied decimal places.
	Discount Value	Number(20)		Value of discount given in this prom type with 4 implied decimal places.
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	Transaction Count	Number(6)	specified by external system	Number of TDETL records in this transaction set
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	specified by external system	ID of current line being processed by input file.
	File Record Counter	Number(10)		Number of records/transactions processed in current file (only records between head & tail)

Invalid Item/Store File:

The Invalid Item/Store File will only be written when a transaction holds an item that does not exist at the processed location. In the event this happens, the relationship will be created during the program execution and processing will continue with the item and store number being written to this file for reporting.

VAT File:

The VAT file will only be written if a particular item cannot retrieve a VAT rate when one is expected (e.g. the `system_options.vat_ind` is on). In this event, a non-fatal error will occur against the transaction and a record will be written to this file and the Reject file.

Reject File:

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will be created by the interface library routines and the detail records will be created using the `write_to_rej_file` function. A reject line counter will be kept in the program and is required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and trailer record and the file line count will be equal to 0.

A final reject file name, a temporary reject file name, and a reject file pointer should be declared. The reject file pointer will identify the temporary reject file. This is for the purposes of restart recovery. When a commit event takes place, the `restart_write_function` should be called (passing the file pointer, the temporary name and the final name). This will append all of the information that has been written to the temp file since the last commit to the final file. Therefore, in the event of a restart, the reject file will be in synch with the input file.

Error File:

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

Assumption: Variable weight UPC's need to be sent in at the transaction level, not the rolled up level that `posupld` usually receives. This is due to the fact that TDETL lines need to source their qty's from the UPC and not from the specified qty input fields.

Assumption: Given the structure of the `var_upc_ean` table, it is only possible to have 100 different records on this table. This function uses this assumption when allocating memory to hold all the records on the `var_upc_ean` table. The logic will be required enhancements to accommodate a larger number of different variable weight upc types if desired.

Assumption: Non-variable weight UPC's can't start with a prefix that exists on the `var_upc_ean` table.

Assumption: All values passed in through variable weight UPC's will be considered to be in the standard unit of measure.

Inbound Work Order Download [powodld]

Design Overview

This program will download inbound work order information, for purchase orders and/or allocations, to a warehouse management system in a standard Retek flat file.

When records are created or updated on the work order tables for an inbound work order, triggers will write an order number to a work order notification table (lif_wo). This batch program will then read through the notification table and create a flat file containing the inbound work order additions/changes/deletions to a standard Retek flat file. A generic conversion download program will then convert the file to the format needed by the Nautilus Work Order Download program.

Scheduling Constraints

Processing Cycle:	N/A
Scheduling Diagram:	This program should run after the ordrev.pc program
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

The Logical Unit of Work for the program will be a single row from the work order notification table.

Anytime a work order is modified, a record containing the corresponding order number is written to the lif_wo staging table. This table will be used to loop through all work orders that have been created or modified. Work orders for a purchase order should only be downloaded to Logistics if the corresponding order has been downloaded. To determine this, check to see if a record exists on ordhead_rev table. If a record exists on ordhead_rev for the order associated with the work order, then download the work order information to Logistics.

Driving cursor:

Fetch records from the lif_wo staging table. Select only work orders which also have a record on ordhead_rev with origin_type = 'V'. Also, to save opening a second cursor later in processing, check to see if each order_no is on wo_prev_dnld.

```
SELECT DISTINCT l.order_no,
               p.order_no
FROM lif_wo l,
     ordhead_rev o,
     wo_prev_dnld p
WHERE l.order_no = o.order_no
     AND l.order_no > TO_NUMBER(:os_restart_order_no)
     AND l.order_no = p.order_no (+)
     AND o.origin_type = 'V'
ORDER BY l.order_no;
```

Program Flow

N/A

Function Level Description

Init()

Initialize restart recovery

Open output file and temp reject file

Call function to format the strings for the output file.

Fetch the vdate.

Write FHEAD record. Call the function that will format FHEAD.

Process()

Loop through the order numbers found on the lif_wo table. An array fetch should be used for performance reasons.

Check if a revision exists for the order number, this indicates that the order is not ready to be downloaded. If no revision exists, then continue with the next work order number.

If a revision exists continue to process.

Check if this is the first time the work order is downloaded, by checking if any records exist on the wo_prev_dnld table.

If the order does not exist on the wo_prev_dnld (ie. no work order information has ever been downloaded for the order), then

Call process_new_po() function.

If the order does exist on the wo_prev_dnld table, then

Call process_existing_po() function.

End looping through the wo order table.

Process_new_po()

Write all work order information with action type = 'A' to the file. This is accomplished by looping through the work order tables (ie. wo_head, wo_sku_loc, wo_wip).

Call write_FDETL()

Call update_snapshot()

Call delete_po()

Process_existing_po()

Call process_deletions()

Call process_additions()

Call delete_snapshot()

Call update_snapshot()

Call delete_po()

Process_deletions()

Any information that does not exist on the work order tables, but exists on the wo_prev_dnld table is considered deleted work order information that needs to be downloaded to the warehouse. Any record found will be considered as an action type = 'D'eleted.

Call Write_FDETL()

Process_additions()

Any information that exists on the work order tables, but not on the wo_prev_dnld table is considered newly 'A'dded work order information that needs to be downloaded to the warehouse.

Call Write_FDETL()

Delete_snapshot()

Delete from wo_prev_dnld table for the processed order.

Update_snapshot()

After detail records have been written to the output file for a specific work order, a new snapshot will be taken for the most recent download of work order information. To take a snapshot of the data use an insert select statement that will insert all work order information for a particular order to the wo_prev_dnld table.

Delete_po()

Delete order number from the lif_wo table, after order is processed.

Format_buffer()

Format fhead - see file layout for default values and lengths

Format fdetl - see file layout for default values and lengths

Format ftail - see file layout for default values and lengths

write_FHEAD()

char *function = "write_FHEAD";

Use fprintf to write to fhead buffer

Set pl_detail_no to 1 – to initiate the counting of total details in file

Increment pl_line_no – to track the total number of records in the file.

write_FDETL()

char *function = "write_FDETL";

Use fprintf to write to fhead record

Increment pl_line_no – to track the total number of records in the file

Increment pl_detail_no – to track how many detail records are written

write_FTAIL()

char *function = "write_FTAIL";

use fprintf to write ftail record

final()

Write the FTAIL record.

Call restart_close

Call restart_file_close – Determines if the temp file is ready to be appended to the final output file.

Close temp file

Remove temp file

I/O Specification

Output File

The output file should be accepted as a runtime parameter at the command line.

Work Order Download:

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	Ten zeroes: 0000000000	ID of current line being processed by input file.
	File Type Definition	Char(4)	INBD	Identifies file as 'Inbound Work Order Download'
	File Create Date	Date	Create date	date file was written by external system
Transaction Detail	File Type Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	Action Type	Char(1)	Action_type	The action being performed on the work order. 'A', 'M', or 'D'

Record Name	Field Name	Field Type	Default Value	Description
	Location (DC)	Char(4)	Prev_wo.location or wo_sku_loc.location	When an item is crossdocked, this field holds the value of the flow-through warehouse. Otherwise it holds the value of the final destination.
	Transaction Date/Time	Char (12) format: YYYYMM DDHHMI	Vdate	sysdate without the seconds
	Purchase Order Number	Char(9)	Prev_wo.order_no or Wo_sku_loc.order_no	This field will hold the order number the work order is associated with. It should be preceded with the letter 'P' to indicate the work order is associated with a purchase order.
	Item ID	Char (16)	Prev_wo.sku or Wo_sku_loc.sku	Valid item identifier for a staple SKU, fashion SKU, or Pack Item
	Dest ID	Number(4)	Prev_wo.location or Wo_sku_loc.location	Unique identifier of the final shipping destination.
	WIP Sequence No.	Number(7)	Prev_wo.seq_no OR Wo_wip.seq_no	Work in Progress Sequence Number
	WO WIP Code	Char (6)	Prev_wo.wip_code Wo_wip.wip_code	WWIP code from codes table
File Trailer	File Type Descriptor	Char(5)	FTAIL	Identifies file record type

Record Name	Field Name	Field Type	Default Value	Description
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	File Record Counter	Number(10)	Determined internally	Number of records/transactions processed in current file (only records between head & tail)

Note: There is a space between fields in the RLS flat file format

Error File

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

N/A

Pre/Post Functionality for Multi-Threadable Programs [prepost]

Design Overview

The Pre/Post module facilitates multi-threading by allowing general system administration functions (such as table deletions or mass updates) to be completed after all threads of a particular program have been processed. A brief description of all pre- or post-processing functions included in this program can be found in the Function-Level Description section.

This program will take three parameters: username/password to log on to Oracle, a program before or after which this script must run and an indicator telling whether the script is a pre or post function. It will act as a shell script for running all pre-program and post-program updates and purges (the logic was removed from the programs themselves to enable multi-threading & restart/recovery).

For example, to run the pre-program script for the ccext program, the following should be entered on the command line:

```
prepost user/password ccext pre
```

Tables affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
COST_SUSP		No	No	No	Yes
COST_SUSP_HEAD		No	No	Yes	No
MOD_ORDER_ITEM HTS		No	NO	No	Yes
ON_ORDER_TEMP		No	No	No	Yes
ORDHEAD		No	No	Yes	No
PERIOD		Yes	No	No	No
POS_MODS		No	No	No	Yes
REQ		Yes	No	No	Yes
REQDETAIL		No	No	No	Yes
WH_STORE_ASSIGN		No	No	No	Yes
SYSTEM_OPTIONS	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	Yes	No

Scheduling Constraints

Processing Cycle:	PHASE ALL (daily)
Scheduling Diagram:	See scheduling flow for description of all pre-post requirements in the daily run.
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A (single threaded)

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

salweek_post():

Updates the last end-of-week date on the SYSTEM_VARIABLES table to today's date after all weekly stock ledger data has been processed.

salmth_post():

Updates the following SYSTEM_VARIABLES columns to reflect the current date's values after all monthly stock ledger data has been processed:

- last_eom_half_no
- last_eom_month_no
- last_eom_date
- next_eom_date
- last_eom_start_half
- last_eom_end_half

- last_eom_start_month
- last_eom_mid_month
- last_eom_next_half_no
- last_eom_day
- last_eom_week
- last_eom_month
- last_eom_year
- last_eom_week_in_half

rpl_pre():

Truncates the following tables before replenishment extracts are performed:

- ORD_TEMP
- ORD_SUGGF
- ORD_SUGG

pcovrl_pre():

Truncates the PRICE_CONFLICT table before new conflicts are found and written.

rplatupd_pre():

Truncates the MC_REJECTIONS table so that it is free to hold new mass change rejections.

rplatupd_post():

Truncates the holding tables REPL_ATTR_UPDATE_ITEM and REPL_ATTR_UPDATE_LOC after their records have been processed.

pctranex_post():

Deletes from the table PRICE_BATCH_TRAN after the price changes it held have been processed.

supmth_post():

Truncates the table SUP_DATA after all daily supplier data records have been rolled up to month level.

sccext_post():

Updates all processed supplier cost change record statuses to 'Extracted'.

rplbld_pre():

Updates replenishment order origins from 'Current System Generated' to 'Past System Generated' before processing. Enables the following triggers:

- RMS_TABLE_OHE_AIUR
- RMS_COL_OHE_STATUS_AUR
- RMS_TABLE_ALH_AIDR
- RMS_TABLE_OSK_AIUR
- RMS_COL_ALD_QA_AUR
- RMS_TABLE_ALD_AIUDR
- RMS_TABLE_OLO_AUR
- RMS_COL_OLO_QTYORDERED_AUR

rplprg_post():

Disables the following triggers:

- RMS_TABLE_OHE_AIUR
- RMS_COL_OHE_STATUS_AUR
- RMS_TABLE_ALH_AIDR
- RMS_TABLE_OSK_AIUR
- RMS_COL_ALD_QA_AUR
- RMS_TABLE_ALD_AIUDR
- RMS_TABLE_OLO_AUR
- RMS_COL_OLO_QTYORDERED_AUR

hstbld_post():

Truncates the holding table MASK_REBUILD after building history records.

posdnld_post():

Clears the POS_MODS table after all records have been downloaded to the POS.

poscdnld_post():

Clears the config_status and loc_grp_status in POS_LOC_GRP. Sets all values of extract_req_ind to 'N'.

Clears the status column in POS_MERCH_CRITERIA.

Sets the status_ind column in POS_STORE to 'N'.

prmext_post():

Updates the promotion item status on the PROMSKU table, setting it to NULL if the promotion item has been extracted successfully or to 'Delete Processed' if the item was marked for deletion before extraction.

edidlprd_post():

This function should be run after the edidlprd program.

It needs to be run as follows from the command line:

```
prepost username/password edidlprd post
```

Deletes old records from the EDI_DAILY_SALES table after they have been processed.

fcstrbld_post():

Truncates the holding table FORECAST_REBUILD after all records have been processed.

vrplbld_post():

Truncates the EDI_ORD_TEMP table after all replenishment orders have been build from the data held there.

fsadnld_post():

Updates the load_sales_ind to 'N' for all records on the appropriate domain table – domain_dept, domain_class, or domain_subclass, where system_options.domain_level = 'D', 'C', or 'S', respectively.

whstrasg_post():

deletes all warehouse store assignment records from the warehouse store assignment table if the assignment date (wh_store_assign.assign_date) is less than or equal to the current date (period.vdate) minus the warehouse store assignment history days (system_options.wh_store_assign_hist_days).

cntrordb_post()

Set the last_cont_order_date on system_variables to vdate.

fifglndn1_post()

If Oracle Financials is being used, delete everything from the fif_receiving table and repopulate it from the if_tran_data table

sub_items_delete_post():

Delete items that are not on repl_item_loc from sub_items_detail and sub_items_head.

policy_enable():

enables or disables policies

truncate_user_sec_table()

truncate the user security tables and prepare for the user product security rebuild, user location security rebuild or user zone security rebuild.

update_security_indicator():

reset the update_sec_xxx_ind to 'N' where xxx represents prd, loc or zon if it is user product security rebuild, user location security rebuild or user zone security rebuild, respectively

tifposdn_post()

truncate tif_explode table

dealcalc_post():

Truncates the deal_sku_temp table.

htsupld_pre()

Truncates the mod_order_item_hts table so that reports will be correct and not include data from previous runs of htsupld.

onordext_pre()

Truncates the on_order_temp table so onordext will not include previous data.

I/O Specification

N/A

Technical Issues

N/A

[primpdep.pc]

Design Overview

The promotion department impact batch program, primpdep.pc, is one of two new batch programs that will calculate the estimated impact for a proposed promotion. Primpdep will call a package for each promotion on promhead where impact_date is not null. The package called in the program, prm_impact_calc.build_dept(), will generate the department-level impact analysis.

Scheduling Constraints

Processing Cycle: This process may be run on-demand, but should be scheduled at a minimum once per day.

Scheduling Diagram: primpku.pc → (must come before) → primpdep.pc

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: promotion

Restart Recovery

Driving Cursor

```
SELECT p.promotion
FROM promhead p,
     v_restart_promotion rv
WHERE p.impact_date <= to_date(:ps_vdate, 'YYYYMMDD')
AND p.status is NOT NULL
AND rv.driver_value = p.promotion
AND rv.driver_name = :ps_restart_driver_name
AND rv.num_threads = :ps_restart_num_threads
AND rv.thread_val = :ps_restart_thread_val
AND p.promotion > NVL(:ps_restart_promotion, -999)
ORDER BY p.promotion;
```

Note: The view, v_restart_promotion, does not exist yet and must be created.

Shared Modules

Prm_impact_calc.build_dept() – New function for calculating the DEPT impact of a proposed promotion.

Function Level Description

All database interactions required and error handling considerations

Init() – Size variables in the fetch for the driving cursor, initialize restart recovery.

Process() – Fetch from the driving cursor. For each id, call PRM_IMPACT_CALC.BUILD_DEPT(). Pass the promotion number and pass the force_recalc_ind = 'N'. Set the promhead.impact_date for the current promotion to null so it will not be picked up during the next batch cycle. After processing array, call restart_force_commit().

Final() – Call restart_close().

Technical Issues

New view should be created for promotions.

[primpsku.pc]

Design Overview

The promotion SKU impact batch program is one of two new batch programs that will calculate the estimated impact for a proposed promotion. Primpsku will call a package for each promotion on promhead where impact_date is not null. The package called in the program, prm_impact_calc.build_sku, will generate the item-zone level impact analysis..

Scheduling Constraints

Processing Cycle: This process may be run on-demand, but should be scheduled at a minimum once per day.

Scheduling Diagram: primpsku.pc → (must come before) → primpdep.pc

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: promotion

Restart Recovery

Driving Cursor

```
SELECT p.promotion
FROM promhead p,
     v_restart_promotion rv
WHERE p.impact_date <= to_date(:ps_vdate, 'YYYYMMDD')
AND p.status is NOT NULL
AND rv.driver_value = p.promotion
AND rv.driver_name = :ps_restart_driver_name
AND rv.num_threads = :ps_restart_num_threads
AND rv.thread_val = :ps_restart_thread_val
AND p.promotion > NVL(:ps_restart_promotion, -999)
ORDER BY p.promotion;
```


Note: The view, v_restart_promotion, does not exist yet and must be created.

Shared Modules

Prm_impact_calc.build_sku() – New function for calculating the SKU impact of a proposed promotion

Function Level Description

Init() – Size variables in the fetch for the driving cursor, initialize restart recovery.

Process() – Call PRM_IMPACT_CALC.BUILD_SKU() for each promotion returned by the driving cursor. After processing array, call retek_force_commit().

Final() – Call retek_close().

Technical Issues

New view should be created for promotions.

Promotion Price Extract and Download [prmext]

Design Overview

The prmext program extracts promotions from the promotion master tables within the Retek system and sends promotion price details to the point of sale system. Additionally, promotional price history is stored for each valid SKU/store combination. When a store is within the POS threshold extraction date, all SKUs on the promotion will be extracted to the store (provided the item is stocked at the store). Additionally, the prmext program has the ability to extract promotion SKU changes throughout the life of a promotion based on the promsku status. The promsku status will provide an indication of SKU details that have not been extracted. (Possible changes are new SKU / deleted SKU / changed promotion price.) The SKU will be re-extracted to stores that are currently active with the given promotion. Stores that are to be extracted will not differentiate between promotion SKU changes since all SKUs will be extracted to the POS, provided the SKU has not been deleted from the promotion.

The promotional retail is stored in the history tables in the local currency of the store and this is also the price that is transmitted to POS. Since the extraction process is performed at a promotion store level, this allows different stores to be effectively on promotion for varying time frames. If the promotion start is within pos_extract_days from tomorrow, the promotion store will be extracted to the point of sale, and a price history record is written with a future action date. The status of the promotion store is updated to extracted. Once all stores on a promotion have been extracted the overall status of the promotion on the header is set to extracted. If the promotion store end date is within pos_extract_days from tomorrow, the regular price will be extracted to the POS and the promotion store status will be updated to completed. Once all store have completed, the overall promotion header status is updated to completed.

For each promotion which is due to start or end within the number of days in pos_extract_days from UNIT_OPTIONS, a POS_MODS row is built containing the details necessary for the POS PLU update for each item (SKU) included in the promotion. Note that if a fashion style has been included in a promotion, it must be expanded to it's component SKUs (sizes and colours).

This program checks overlap with price changes if an overlap is found, it does not insert price_hist, but still inserts pos_mods, with the price fetched from the price change tables.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
UNIT_OPTIONS	No	Yes	No	No	No
PERIOD	No	Yes	No	No	No
PROMHEAD	Yes	Yes	No	Yes	No
PROMSTORE	Yes	Yes	No	Yes	No
PROMSKU	Yes	No	No	No	No
WIN_SKUS	Yes	Yes	No	No	No
RAG_STYLE	Yes	Yes	No	No	No
RAG_SKUS	Yes	Yes	No	No	No
PACKHEAD	Yes	Yes	No	No	No
POS_MODS	No	No	Yes	No	No
PRICE_HIST	No	No	Yes	No	No
V_RESTART_STORE	No	Yes	No	No	No
PRICE_SUSP_HEAD	No	Yes	No	No	No
PRICE_SUSP_DETAIL	No	Yes	No	No	No

Restart Recovery

```
SELECT ph.promotion,
       ph.event,
       TO_CHAR(ps.start_date, 'DDMMYYYY'),
       TO_CHAR(ps.end_date, 'DDMMYYYY'),
       ps.store,
       0,          /* promotion action type */
       decode(ps.extract_status, 'E', 1, 0),
       ph.rowid,
       ps.rowid
FROM v_restart_store rv,
     promstore ps,
```

```

        promhead ph
WHERE ph.promotion = ps.promotion

        AND ps.start_date<= to_date(:vdate,'DDMMYYYY')+pos_extract_days
        AND ph.status in ('A','E')
        AND nvl(ps.extract_status, 'E') = 'E'
        AND rv.driver_value = ps.store
        AND rv.driver_name = :ora_restart_driver_name
        AND rv.num_threads = :ora_restart_num_threads
        AND rv.thread_val = :ora_restart_thread_val
        AND (ps.promotion > NVL(:ora_restart_promotion, -999) OR
              (ps.promotion = :ora_restart_promotion AND
               (ps.store >= :ora_restart_store)))
UNION ALL
SELECT ps.promotion,
        ph.event,
        TO_CHAR(ps.start_date, 'DDMMYYYY'),
        TO_CHAR(ps.end_date, 'DDMMYYYY'),
        ps.store,
        1,          /* promotion action type */
        0,          /* set to not extracted */
        ph.rowid,
        ps.rowid
FROM v_restart_store rv,
     promstore ps,
     promhead ph
WHERE ph.promotion = ps.promotion

```

```
AND ps.end_date <= to_date(:vdate,'DDMMYYYY')+ :pos_extract_days
AND (ps.extract_status = 'E'
OR (ps.extract_status is NULL AND ps.end_date = ps.start_date))
AND rv.driver_value = ps.store
AND rv.driver_name = :ora_restart_driver_name
AND rv.num_threads = :ora_restart_num_threads
AND rv.thread_val = :ora_restart_thread_val
AND (ps.promotion > NVL(:ora_restart_promotion, -999) OR
(ps.promotion = :ora_restart_promotion AND
(ps.store >= :ora_restart_store)))
ORDER BY 1,5,6;
```

Scheduling Constraints

Processing Cycle: PHASE 1 – (daily)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: Post processing for prmext resets all promotion SKU status after the SKU has been extracted (or re-extracted) to a store. Since stores have various start and stop dates and promotion maintenance can occur at any point during a promotion, the prepost program resets the change status on promsku to ensure that the same change is not forwarded down to POS more than once.

Threading Scheme: STORE
V_restart_store

Program Flow

N/A

Shared Modules

GET_SYSTEM_IND: fetches the merchandise type for the SKU to be processed from the desc_look table.

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Promotions Purge [prmprg]

Design Overview

All old promotions are purged when the number of months since the end_date of the promotion is greater than prom_history_months from UNIT_OPTIONS. Also, any promotion that is in deleted 'D' or canceled 'C' status will be purged from the system. Rejected promotions that remain in the system beyond the threshold number of promotion reject days will also be purged.

The following database tables are accessed by this program:

TABLE	SELECT	INSERT	UPDATE	DELETE
UNIT_OPTIONS	Yes	No	No	No
PERIOD	Yes	No	No	No
PROMHEAD	No	No	No	Yes
PROMSTORE	No	No	No	Yes
PROMSKU	No	No	No	Yes
PROM_THRESHOLD_HEAD	No	No	No	Yes
PROM_THRESHOLD_DEPT	No	No	No	Yes
PROM_THRESHOLD_SKU	No	No	No	Yes
PROM_MIX_MATCH_HEAD	No	No	No	Yes
PROM_MIX_MATCH_BUY	No	No	No	Yes
PROM_MIX_MATCH_GET	No	No	No	Yes
RESTART_CONTROL	Yes	No	No	No
SA_TRAN_DISC_REV	No	No	No	Yes
SA_TRAN_DISC	No	No	No	Yes
FIXED_DEAL	No	No	No	Yes
FIXED_DEAL_DATES	No	No	No	Yes
PROMSKU_IMPACT	No	No	No	Yes
PROMDEPT_IMPACT	No	No	No	Yes

Scheduling Constraints

Processing Cycle:	PHASE 4 (monthly)
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

```
SELECT p.promotion
      FROM promhead p
     WHERE (:prom_history_months <
            (MONTHS_BETWEEN(to_date(:vdate),p.end_date))
            AND p.status = 'M')
            OR p.status in ('D', 'C')
            OR ((p.start_date - :prom_reject_days) > to_date(:vdate)
            AND p.status = 'R');
```

Program Flow

N/A

Shared Modules

N/A

Technical Issues

N/A

Item Level Promotions Explode [prmxpld]

Design Overview

The promotion explode module expands records from the promdept table to the promsku table. The program will be run prior to the prmext module that will extract records from the promsku table to be written to price_hist and posmods. Once a promotion has been expanded to SKU level, changes will not be allowed at the department level. Since individual SKUs could be added, or changed after expansion and due to complications that could arise due to reclassification, maintenance functions will only be supported at the SKU level.

The following tables are accessed by this program:

TABLE	SELECT	INSERT	UPDATE	DELETE
PROMDEPT	Yes	No	No	No
PROMSKU	Yes	Yes	No	No

Scheduling Constraints

Processing Cycle: PHASE 1 (daily)

Scheduling Diagram: The prmxpld must be run daily prior to the prmext module

Pre-Processing: N/A

Post-Processing: Post process is required to reset the status of the promdept records from 'AI' to null

Threading Scheme: V_restart_promo (this program can be multi-threaded by active promotion)

```
select promotion
from   promhead
where  status_code = APPROVED
and    ph.start_date -
system_variables.pos_extract_days <=
TOMORROW
and    ph.end_date > TODAY
```

Restart Recovery

The logical unit of work for this promotion explosion module will be the expansion of a promotion subclass combination. All rows on the promdept table will be expanded to the subclass level by the on-line promotion entry conversation. Once all the SKUs have been expanded in a subclass a commit threshold will be checked. If the commit counter has reached the threshold level, a commit will be issued (always at a subclass boundary). The restart string will therefore consist of the promotion, department, class and subclass last processed prior to the commit. A reasonable commit frequency for this module would be 1000 records.

```
select  promotion,
        dept,
        class,
        subclass
        to_char(promotion)||to_char(dept)||to_char(class)||to_char(subclass)
from    promdept pd,
        v_restart_active_promo rv
where   pd.promotion = rv.driver_value
and     rv.thread_val = THREAD
and     rv.num_threads = NUN_THREADS
and     pd.change_type != 'EX'
and     pd.status_code = 'AI'
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

```
init()
initialize restart recovery
```

```
process()
open driving cursor
loop for all promotion dept/class/subclass combinations to be processed
    fetch from driving cursor
    call fill_promsku_array()
    if commit threshold has been reached
        call insert_promsku()
    end if
endloop

fill_promsku_array

declare cursor to get all SKUs in the given dept/class/subclass from all item types
from the

rag_style, win_skus, and packhead tables, not including any SKUs or styles that
are already on the promsku table for the current promotion.

open SKU cursor

loop for all SKUs in the dept/class/subclass
    fetch from SKU cursor

fill promsku array with all SKUs and Styles

increment commit threshold counter

endloop

insert_promsku()

insert into the promsku table all records from the promsku array
```

Technical Issues

N/A

Promotions Download to Output File [promdnld]

Design Overview

The promdnld.pc program will be used to send down promotional information for the mix_match, threshold, and multi_unit promotional types. This information will exist on the pos_prom_detail table that is populated in prmxext.pc. This file will assist the POS in calculating the price of a sale as well as allow it to calculate the best price to give the customer when multiple promotions are at play.

Because the pos_prom_detail table only will contain information for the mix_match, threshold, and multi_unit promotional types, this program, this program will only need to be ran when the multi_prom_ind on system_options is set to 'Y'. Therefore, in the init() function, if this indicator is retrieved as 'N', exit the program normally.

This program will also be written in the Retek standard file format. The format used for this program will be: FHEAD, FDETL, FTAIL.

Scheduling Constraints

Processing Cycle: PHASE 4 (daily)

Scheduling Diagram: This program is run towards the end of the batch run when all pos_mods records have been created for the transaction day.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: Threading done by store

Restart Recovery

Restart/recovery for this program is set up at the store level. Likewise, threading is done by store using the v_restart_store view to thread properly.

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

Program Flow



Shared Modules

N/A

Function Level Description

Init

This function initializes restart/recovery for this program. It also opens the output file, retrieves system variables and calls a function to size the arrays used in this program.

Process

This function drives the processing of the program. The driving cursor is fetched here which retrieves all the records from pos_prom_detail where the pos_prom_detail.store value is greater than zero. Once the records are fetched, the process_prom_detail() function is called to perform processing on them. After each record is moved to the output file, pos_prom_delete() will remove the record from the table. Restart/Recovery and committing of records is also performed here.

Process_prom_detail

This function will prepare records for insert into the output file. This program used the Retek standard file format FHEAD, FDETL, FTAIL.

Pos_prom_delete

This function deletes the current record from the pos_prom_detail table.

Final

This function will finish restart/recovery logic, close the output file and delete the temporary output file used while the program processes.

Init_format_strs

This function formats the strings for the FHEAD, FDETL, and FTAIL records in the output file.

size_arrays

This function initializes the size of the array used for the driving cursor fetch the size of the restart max counter on restart_control.

I/O Specification

Promotion_detail file

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)	Sequential number Created by program.	ID of current line being created for output file.
	File Type Definition	Char(4)	PRMD	Identifies file as 'PROM Download'
	File Create Date	Char(14)	create date	current date, formatted to 'YYYYMMDDHH24MISS'.
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	Sequential number Created by program.	ID of current line being created for output file.
	Location Number	Number(4)	store identifier	Store identifier from the Retek system.
	Promotion Number	Number(4)	promotion number	Promotion number for SKU from Retek promotion tables.

Record Name	Field Name	Field Type	Default Value	Description
	Transaction Type	Char(6)	promotion transaction type	Transaction type indicating the type of promotion taken from the code_detail table where code_type = 'PRMT'.
	Start time	Char(6)	start time	start time of the promotion, formatted to 'HH24MISS' from Retek promotion tables.
	End time	Char(6)	end time	end time of the promotion, formatted to 'HH24MISS' from Retek promotion tables.
	Mix Match Number	Number(4)	mix match number	mix match number for SKU from Retek promotion tables.
	Threshold Number	Number(4)	threshold number	threshold number for the SKU from Retek promotion tables.
	Apply To	Char(1)	'C' for Compound or 'E' for Exclusive	Indicates if the promotion can be used with other promotions. From Retek promotion tables.
	Buy Type	Char(1)	'A' for All, 'U' for Any	Indicates how many of the prom mix match buy SKU's must be purchased to qualify for the promotion. From Retek promotion tables.

Record Name	Field Name	Field Type	Default Value	Description
	Buy Amount	Number(20)	buy amount	The amount that has to be purchased to qualify for the promotion. From Retek promotion tables.
	Get Type	Char(1)	'P' for Percent off, 'A' for Amount off, or 'F' for fixed amount.	Indicates how to lower the price for the prom mix match get SKU's. From Retek promotion tables.
	Get Amount	Number(20)	get amount	The amount or percent off of the Get SKU(s) or the amount the SKU(s) are on promotion for. From Retek promotion tables.
	Threshold Amount	Number(20)	threshold amount	The threshold level needed to be reached to qualify for the promotion. From Retek promotion tables.
	Discount Amount	Number(20)	discount amount	The amount of discount when the threshold amount is reached. From Retek promotion tables.
	Discount Type	Char(1)	Discount type	Discount type indicator.
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	Sequential number Created by program.	ID of current line being created for output file.

Record Name	Field Name	Field Type	Default Value	Description
	File Record Counter	Number(10)		Number of records/transactions processed in current file (only records between head & tail)

Technical Issues

N/A

Automatic Shipment Extraction [rcvext]

Design Overview

This module checks the system level indicator, `system_options.auto_shipment_ind`, to determine if shipments should be automatically created. If `auto_shipment_ind` is set to yes, the program will process all approved orders where the supplier is not an ASN supplier, the `not_before_date` for the order is within `system_options.auto_shipment_days` of today's date and shipments have not already been created for the order. For each order that meets the criteria, shipments and shipment detail records will be created.

If Nautilus is turned on, then this program will not create shipments destined to warehouses. A parameter (`insert_flag`) will be added to the shipment insert package call. The returned true or false parameter will be checked before the calling the insert into `shipsku` function. If the parameter is set to 'True', then call the insert into `shipsku` function. If the parameter is set to 'False', then do not call the function.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ORDHEAD	Yes	Yes	No	No	No
PERIOD	No	Yes	No	No	No
SUPS	Yes	Yes	No	No	No
SHIPMENT	Yes	Yes	Yes	No	No
SHIPSKU	Yes	Yes	Yes	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE 1 (daily)

Scheduling Diagram: This program must run before the external system files for receiving are uploaded. It will include the orders created during the previous day's batch cycle as well as any orders that were created on-line during the current business day.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

```
SELECT ordhead.order_no,  
       TO_CHAR(ordhead.not_before_date, 'DDMMYYYY'),  
       TO_CHAR(ordhead.supplier),  
       ordhead.qc_ind  
FROM ordhead,  
       sups  
WHERE ordhead.status_ind = 10  
      AND (ordhead.not_before_date - :auto_shipment_days) <=  
          to_date(:vdate)  
      AND ordhead.supplier = sups.supplier  
      AND sups.edi_asn = 'N';
```

Program Flow

N/A

Shared Modules

SHIPMENT_INSERT is called to insert shipment records.

SHIPSKU_INSERT is called to insert shipsku records.

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

P.O. Receipt Transactions Upload [rcvupld]

Design Overview

The purpose of this batch module is to accept receipt details from an external system. The receipt transactions will provide feedback on orders existing in the Retek system, and will cross-reference shipments if they exist. The receipt detail information can be processed using any of the supported item types (i.e. SKU, UPC, or VPN) within the RMS system. It also processes receiver unit adjustment, as well as accepts item that does not exist on the purchase order.

The following functions will be performed for each item received:

- Create/update shipment and shipment/SKU records
- Create/update order/SKU and order/SKU/location records
- Update order header status to complete if all units received against the order
- Create item/location relation for receiving location (if it doesn't exist)
- Update perpetual inventory
- Update average cost of item at receiving location (if stock on hand not = 0 or negative)
- Update unit cost through package call to `RECEIVE_SQL.ITEM` if the cost change is a base cost change (supplier is the primary supplier for the item)
- Write stock ledger financial transactions
- Update the snapshot stock on hand quantity for received item/location if a stock count is in progress
- Create transfers based on allocations tied to the order
- Update OTB table's receipt amount
- Write supplier data record for vendor analytics
- Update `contract_detail` and `contract_header` if PO is for a contract
- Write `ticket_request` if item is received or allocated into store and if the ticket is generated at PO receipt time
- Write to variable `asn_exists` if the supplier is using "Evaluated receipt settlement" as payment process method and a matched shipment is found for the receipt

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ALLOC_HEADER	Yes	Yes	No	Yes	No
ALLOC_DETAIL	Yes	Yes	No	Yes	No
PACKWH	Yes	No	Yes	Yes	No
ORDLOC	No	Yes	Yes	Yes	No
ORDSKU	No	No	Yes	Yes	No
ORDHEAD	No	No	No	Yes	No
OTB	No	No	No	Yes	No
CONTRACT_HEADER	No	Yes	No	Yes	No
CONTRACT_DETAIL	No	Yes	No	Yes	No
RAG_SKUS_ST	Yes	Yes	Yes	Yes	No
RAG_SKUS_WH	Yes	Yes	Yes	Yes	No
WIN_STORE	Yes	Yes	Yes	Yes	No
WIN_WH	Yes	Yes	Yes	Yes	No
SHIPMENT	No	Yes	Yes	Yes	No
SHIPSKU	No	Yes	Yes	Yes	No
STAKE_SKU_LOC	No	No	No	Yes	No
SUP_DATA	No	No	Yes	Yes	No
TRAN_DATA	No	No	Yes	No	No
TSFHEAD	Yes	Yes	Yes	Yes	No
TSFDETAIL	Yes	Yes	Yes	Yes	No
V_SKU_INFO	Yes	Yes	No	No	No
WIN_SKUS	Yes	Yes	No	Yes	No
RAG_SKUS	Yes	Yes	No	Yes	No
PACKHEAD	Yes	Yes	No	Yes	No

Scheduling Constraints

Processing Cycle: PHASE 2 (daily)

Scheduling Diagram: This program must run after tsfparse.pc. This program will likely be run at the beginning of the batch run during the POS polling cycle, or possibly at the end of the batch run if pending warehouse transactions. It can be scheduled to run multiple times throughout the day, as WMS or POS data becomes available.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: STORE and WH
Driven by distinct files by location, or consolidated receipt files.

Restart Recovery

The logical unit of work for the receiving module is the discrete receipt transaction. Each receipt will be identified by the vendor ASN number (or shipment number if non-EDI), the Retek order number and a unique transaction set number generated by the external system. This receipt transaction will be defined as the logical unit of work. If any portion of the processing for the complete receipt transaction fails, the entire receipt must be re-processed.

A savepoint will be issued prior to processing a new receipt. If any record within the transaction fails, the whole transaction will be rolled back to the most recent savepoint. This way, the successfully processed transactions will remain posted to the database but not yet committed.

To prevent excessive rollback space usage, intermittent commits will be issued based on a commit counter. The recommended commit counter setting is 10000 records (subject to change based on experimentation). The commit counter is based on actual records processed, not overall transactions, nor the number of writes to the database, since the database interactions will be a constant multiplier of the commit counter. A receipt transaction cannot be committed to the database until it is complete so the commit counter is viewed as a minimum threshold, that once reached, will force a commit after the completion of the current receipt transaction.

Error handling will be based on the logical unit of work also. If a given record within a receipt transaction fails, that error will be posted to the standard error log for the batch module. If the error is of a non-fatal type, all subsequent detail records within the receipt will continue to be processed and any errors noted will continue to be posted. After processing all errors for the transaction, the entire receipt will be rejected to a runtime specified rejection file. If a fatal error is encountered, the file pointer at the time of the last commit will have been posted to the bookmark and all transactions from the last commit will be rolled back. Processing will commence with from the saved file position.

Shared Modules

RECEIVE_SQL.ITEM: Package referenced to perform all receipt logic, including

- Create/update shipment and shipment/SKU records
- Create/update order/SKU and order/SKU/location records
- Update order header status to complete if all units received against the order
- Create item/location relation for receiving location (if it doesn't exist)
- Update perpetual inventory
- Update average cost of item at receiving location (if stock on hand not = 0 or negative)
- Update unit cost through package call to RECEIVE_SQL . ITEM if the cost change is a base cost change (supplier is the primary supplier for the item)
- Write stock ledger financial transactions
- Update the snapshot stock on hand quantity for received item/location if a stock count is in progress
- Create transfers based on allocations tied to the order
- Update OTB table's receipt amount
- Write supplier data record for vendor analytics
- Update contract_detail and contract_header if PO is for a contract
- Write ticket_request if item is received or allocated into store and if the ticket is generated at PO receipt time
- Write AP_tran if the supplier is using "Evaluated receipt settlement" as payment process method and a matched shipment is found for the receipt

The following are called from the RECEIVE_SQL package and are thus indirect calls.

STOCK_LEDGER_SQL.TRAN_DATA_INSERTS: Package referenced by RECEIVE_SQL package to perform the stock ledger transaction inserts for receipt of goods.

NEW_STAPLE_LOC, NEW_FASHION_LOC, NEW_PACK_LOC: These stored procedures are used to create item/location relationships for locations that are to receive goods.

AUTO_TRANSFER_WH: Function in the RECEIVE_SQL package, called when performing a warehouse receipt to create transfers automatically based on allocations tied to the order.

Function Level Description

init()

declare structure array for shipsku

initialize restart recovery

open input file (receipts)

- file should be specified as input parameter to program

open reject file (as a temporary file for restart)

- file should be specified as input parameter to program

get vdate from period table

set application image array - save the line counter

process()

loop

read record from input file

if ('FTAIL') Exit Loop

```
if ( 'THEAD')
    set savepoint and save current file pointer position
    get transaction header record details
validate_THEAD()
reset detail count
process_THEAD()
end if

loop
    read record from input file
    if ( 'TTAIL') Exit Loop
    if ('TDETL')
        validate_TDETL()
process_TDETL()
        write detail transaction to shipsku array
    end if

    if ( detail count = max array count )
        resize array structures for shipsku array
        increase max array count
    end if
    increment detail count
end loop

if ( no errors )
post_receipts()
```

if (non-Fatal Error encountered)

reject_record - call write error and pass file pointer as of last savepoint and
current file pointer

rollback transaction

end if

if (transaction count > max commit count)

restart file commit

- save the current input file pointer position

- save the line counter in restart image

end if

end loop

restart commit final

validate_THREAD()

validate supplier number and get supplier edi_ASN indicator

validate order against supplier, order number, location

if (order number does not exist) Error

if (Retek shipment)

validate shipment exists

if (shipment does not exist) Error

else

validate Vendor Ship / Order No

if (shipment does not exist)

create shipment / shipskus for all outstanding qty at order_location

end if

end if

perform insert of shipskus as array insert

validate_TDETL()

if (Item Type = 'UPC')

select sku from upc_ean based on the upc and supplement

if (upc does not exist)

write non-Fatal Error (upc not found)

end if

end if

process_TDETL()

- get sku system indicator and merchandise hierarchy

if (system indicator does not exist)

write non-Fatal Error (sku not found)

end if

call RECEIVE_SQL.NO_ORDER package function

call RECEIVE_SQL.ITEM package function

(see design specification for RECEIVE_SQL)

update_shipsku()

updates the shipsku table with the quantity received from the input file; insert a record if none exists

create_shipment()

get next shipment# and insert into shipment and shipsku tables

ON Fatal Error

rollback to last physical commit point

Exit Program

ON Non-Fatal Error

rollback to last savepoint

write out complete receipt transaction to the reject file, pass file pointer at last savepoint and current file pointer

I/O Specification

Input File

The input file should be accepted as a runtime parameter at the command line.

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)	RCPT	Identifies file as 'Receipt'
	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	Date	specified by external system	date the transfer was created in external system
	Location Type	Char(2)	ST - store WH - warehouse	specifies the type of receiving location
	Location Value	Char(4)	location identifier	specifies the receiving location id number
	Adjustment Flag	Char(1)	'Y' – Adjustmen t 'N' – Not Adjust.	Not used for Transfers – will always be 'N' (Used for receiver error adjustment for common receiving layout).

Record Name	Field Name	Field Type	Default Value	Description
	Order Number	Char(8)	Retek order number	specifies the Retek order cross reference
	Shipment Number	Char(10)	Retek shipment number	specifies the Retek shipment cross reference
	Supplier Identifier	Char(10)	Retek supplier number	specifies the Retek shipment cross reference

Record Name	Field Name	Field Type	Default Value	Description
	Vendor ASN	Char(15)	vendor shipment number reference	reference to vendor shipping document
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
	Detail Sequence Number	Char(6)	specified by external system	sequential number assigned to detail records within a transaction
	Item Type	Char(3)	UPC SKU VPN	item type will be represented as a UPC or SKU or VPN
	Item Value	Char(30)	item identifier	the id number of a SKU or UPC(or VPN)
	Supplement	Char(5)	supplemental identifier	used to further specify the id of an UPC item
	Carton	Char(20)	carton number	unique identifier of each carton associated with a shipment.
	Inventory Status	Char(2)	inventory status	used to indicate the type of non-salable merchandise transferred in an 'NS' transfer

Record Name	Field Name	Field Type	Default Value	Description
	Receipt Quantity Sign Flag	Char(1)	'P' – positive	Always positive for transfer (file layout common to receiving which can have pos. or neg. adjustments)
	Receipt Quantity	Char(12)		number of units received of the given item
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

Reject File

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will need to be created by the receiving module and a reject line counter will be required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and trailer record and the file line count will be equal to 0.

The reject filename should also be specified as a runtime parameter.

Error File

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

N/A

Receipts SQL Loader [receipts_upload.ctl]

Design Overview

The following program is a SQL Load script to load the Nautilus Receipts upload file, receipt_upload.dat, into a staging table: lif_receipts

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: SQL loading must occur after Nautilus creates the receipts flat file, and it must occur before lifrcvup.pc.

Pre-Processing: N/A

Post-Processing: Sqlldr '.bad' files will be written with any data that cannot be inserted into the staging tables. Also, '.log' files will be created to view the results of the Sql load.

Threading Scheme: N/A

Program Flow

N/A

Function Level Description

N/A

I/O Specification

SQL Load the Logistics' receiving upload file using the following control file (ie. receipts_upload.ctl):

Fields are specified by position in file, not by using delimiters.

LOAD DATA

INFILE 'receipts_upload.dat'

INTO TABLE lif_receipts

(location	POSITION(1:4)	INTEGER EXTERNAL,
PO_nbr	POSITION(14:22)	CHAR,
Item_id	POSITION(24:39)	CHAR,
Unit_qty	POSITION(41:49)	NUMBER
Tran_type	POSITION(51:51)	CHAR,
Date	POSITION(53:64)	CHAR,
Receipt_nbr	POSITION(66:74)	INTEGER
EXTERNAL,		
ASN_nbr	POSITION(76:95)	CHAR,
Dest_id	POSITION(97:100)	INTEGER EXTERNAL
Carton_id	POSITION(123:142)	CHAR)

To invoke SQL Loader user the following command line: sqlldr
userid/password@database control=receipts_upload.ctl log=receipts_upload.log

Technical Issues

N/A

Item Reclassification [reclsdly]

Design Overview

The Item Reclassification batch program is executed in order to reclassify items from one department, class or subclass to another. This reclassification of items into different merchandise hierarchy level is initiated or requested online in the item reclassification dialog, with an effective date specified. This program reads in the reclassification requests that are effective the following day, and for each item being reclassified, the following functions are executed :

- Check if item is forecastable and if it is, then check for the existence of a domain. If the item is forecastable and no domain association to the new merchandise hierarchy level exists, reject the item (i.e the item will not be reclassified). This is only done for Staple and Fashion since pack items are not forecastable.
- Check if the item has UDA (user defined attributes) defined for all required UDA's of the new merchandise hierarchy level, if not, reject the item
- Check if the item has UDA defined for all UDA's that have default set up for the new merchandise hierarchy level, if not, a warning message is written to the report, but item will still be reclassified
- update desc_look.dept for the item
- update the appropriate item table, e.g. win_skus, with the new merchandise hierarchy
- insert records into pos_mods with a transaction code of 13 (item reclassification) for each item/store combination
- insert records into tran_data with transaction types of 34 (reclassification in) and 36 (reclassification out) for each sku/store combination
- if the reclassification causes a change of domains for the item, the item store tables are updated, setting the last_sales_export_date to NULL. A NULL will result in all the item store's sales history to be downloaded during the sales download process to the external forecasting system. This is required because of the domain change.
- If item exists on an order with status = approved, OTB table is updated to transfer cancel_amt, approved_amt and received_amt from the old merchandise hierarchy level to the new merchandise hierarchy level
- If an item is reclassified, the product securities of the item are then updated.

Scheduling Constraints]

Processing Cycle: Daily

Scheduling Diagram: This module is scheduled to run daily during Phase 4 of the batch schedule. The requirement is for all the batch modules that update inventory and retail / cost prices to be completed before this module can run.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: If desired, this program can be threaded by reclass_no to process several reclassifications at once.

Restart Recovery

The logical unit of work for the reclassification is reclass_no/sku. The restart commit counter will need to be carefully determined by each client according to the number of sku/store combinations that will be affected by the reclassification since processing is done inside the packages rather than directly from the batch program. Large reclassifications with thousands of skus held at many stores need smaller commit counters to avoid reprocessing large amounts of data in the event of program failure. Small reclassifications affecting just a few sku/store combinations can have much larger commit counters since fewer rows will be inserted into the database each time a sku is processed.

The driving cursor should look like the following:

```
SELECT ROWIDTOCHAR(rs.rowid),  
  
       rs.reclass_no,  
  
       rs.sku,  
  
       rh.to_dept,  
  
       rh.to_class,  
  
       rh.to_subclass,  
  
       dl.system_ind,  
  
       dl.dept,  
  
       dl.class,  
  
       dl.subclass  
  
FROM v_restart_reclass rv,
```

```

        reclass_sku rs,
        reclass_head rh,
        desc_look dl
WHERE rh.reclass_no = rs.reclass_no
      AND rh.reclass_date <= :ov_vdate
      AND rs.sku = dl.sku
      AND rv.driver_value = rh.reclass_no
      AND rv.driver_name = :ora_restart_driver_name
      AND rv.num_threads = :ora_restart_num_threads
      AND rv.thread_val = :ora_restart_thread_val
      AND (rh.reclass_no > NVL(:ora_restart_reclass_no, -999) OR
            (rh.reclass_no = :ora_restart_reclass_no AND
             rs.sku > :ora_restart_sku))
ORDER BY rs.reclass_no,
         rs.sku;

```

Shared Modules

Reclass_Sql.Item_Process:

This function performs all the processing necessary to reclassify an item

Deal_Sql.Insert_Deal_Sku_Temp:

This function inserts a record into the 'Deal_Sku_Temp' table for each reclassified sku.

Function Level Description

Restart/Recovery for table-based processing should be implemented for this program.

declarations

declare a structure of arrays to read in multiple rows from the driving cursor

init()

initialize restart/recovery

declare a structure of arrays to read in multiple rows from the driving cursor

set up arrays to receive fetches in table_process() – memory should be dynamically allocated so that the cursor fetches arrays the size of max counter on restart_control (g_l_restart_max_counter)

get the current date

process()

open the driving cursor

while(1)

fetch g_l_restart_max_counter records at a time into the structure, exiting the loop when all rows are processed

for each reclass_head, determine if the new hierarchy is associated to a domain if the system forecast indicator is set to Yes. If not, pass this to Reclass_Sql.Item_Process where the item will be rejected if it is forecastable. Also, get the domain of the new merchandise hierarchy. This domain will be passed to Reclass_Sql.Item_Process to compare against the domain of each item's old merchandise hierarchy if the item is forecastable. If the domains are different, update the item store tables last_sales_export_date.

for each reclassified item

if the reclass_no of this record is different from that of the last record, delete the record with the old reclass_no from reclass_head

call Reclass_Sql.Item_Process to reclassify the item

call DEAL_SQL.INSERT_DEAL_SKU_TEMP to insert each reclassified item into the 'DEAL_SKU_TEMP' table.

Insert all approved orders for each reclassified item into the 'DEAL_CALC_QUEUE' table.

call restart_commit function to execute standard restart/recovery commit logic

delete all records in the structure from the reclass_sku table

update product securities of the item.

end while(1)

close the driving cursor

process_security()

deletes all the records in sec_group_prod_matrix table where sku equals reclassified sku or sku belongs to the reclassified style or style equals reclassified style

deletes records in sec_user_prod_matrix tables where sku equals the reclassified SKU or sku belongs to the reclassified style

updates the sec_user_prod_matrix table

get_order_numbers(char * is_fetched_sku)

- finds all order numbers which are related to the passed sku on the ORDSKU table.
- calls the 'update_deal_calc_queue' function.

Update_deal_calc_queue(char is_order_number[][NULL_ORDER_NO], long il_curr_recs)

- calls the 'order_exists' function.
- inserts the passed order numbers with the 'recalc_all_ind', 'OVERRIDE_MANUAL_IND', and 'ORDER_APPR_IND' fields set to 'N' into the 'deal_calc_queue' table.

Call_insert_deal_sku_temp(char * is_sku, char * is_supplier, char * is_origin_country_id)

- calls the 'DEAL_SQL.INSERT_DEAL_SKU_TEMP' package function.

Get_supplier_origin_country_id(char * is_sku)

- fetches all suppliers and country_origin_ids for the passed sku from the 'ITEM_SUPPLIER' and 'ITEM_SUPP_COUNTRY' tables.

Order_exists(char * is_check_this)

- checks to see whether or not the passed order number already has an entry in the 'DEAL_CALC_QUEUE' table.

final()

call restart_close to finish restart/recovery

ON Fatal Error

exit function with -1 return code

Technical Issues

N/A

Item Requisition Extraction [reqext]

Design Overview

Reqext (Item Requisition Extraction) handles automatic replenishment of items from warehouses to stores. It cycles through every item-store combination that is set to be reviewed on the current day, and calculates the quantity of the item that needs to be transferred to the store (if any). In addition, it distributes this Recommended Order Quantity (ROQ) over any applicable alternate items associated with the item. The program then takes this information and either creates new transfer line items or adds to existing ones.

Alternate items are either simple packs or substitute items. Simple packs are sellable and orderable packs that contain only a single item, such as a six-pack of cola or twelve-pack of socks. Substitute items are items predefined to be interchangeable with the item being replenished (referred to as the master item).

When an item is set up to use simple packs (designated by an indicator on the REPL_ITEM_LOC table), the ROQ must be distributed among these packs according to desirability. If a master item has no simple packs associated with it, it will be requested as itself. If there is only one pack associated with the item (referred to as the primary simple pack), then there is no distribution needed – the item will be transferred in this simple pack, since the cost per item for a pack is always less than that of an individual item. If multiple simple packs can be substituted for an item, then the distribution of the ROQ over these packs is determined by comparing the packs' relative sales history. Replenishing an item through multiple simple packs can have a severely negative effect on the performance of this program! Because the pack distribution depends on access to the huge sales history tables (PACKSTORE_HIST, RAG_SKUS_ST_HIST, WIN_STORE_HIST), it is not recommended that many items be placed on replenishment through multiple simple packs. Whenever possible, it is better to assign a primary simple pack to the item, since this does not require distribution calculation.

If an item is not set up to use simple packs, the program will see if any substitute items are associated with it. If there are no substitute items associated with the master item, it will be transferred alone. If there are substitute items, they will be fetched into a list and the master item placed at either the head or tail end of the list, depending on the fill priority (set on the SUB_ITEMS_HEAD table). The priority determines which items are transferred first.

No matter what type of alternate items (if any) are used, the program will account for availability when building transfer line items. For simple packs, the share of ROQ allocated to each pack may be decreased or increased if the source warehouse has a shortage of some packs but a surplus of others. For substitute items, transfer quantities are prorated by calculating the ratio of total availability to total need, and items are transferred in order of priority until all need is filled or until no stock is available.

Once the transfer quantity of an item has been calculated, the transfer line item is posted to the database if both the actual quantity to transfer is greater than zero, and the replenishment order control indicator for the item-store combination is either Automatic or Semi-Automatic. If it is Manual, a record will be written to another table (REPL_RESULTS) for reporting purposes. If the system-level All Replenishment Results indicator is set to "Yes", all line items will be written to REPL_RESULTS, even if the quantity to order is zero. Whenever a transfer line item is placed, the appropriate item-location table (RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH and/or PACKWH) is updated to reflect the fact that stock is now reserved for transfer at the warehouse and expected at the store.

Scheduling Constraints

Processing Cycle: PHASE 3

Scheduling Diagram: Rplatupd, rpladjf and rpladjf need to run before reqext so that all replenishment calculation attributes are up to date. Posupld, tsfoupld, tsfiupld, ctniupld, and rcvupld need to run before reqext so that all stock information is up to date. Rplext should run after reqext, since the ROQ for a warehouse is influenced by any transfers created.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: DEPT

Restart Recovery

The logical unit of work is item, source warehouse. The driving cursor is ordered by item, source warehouse, order control indicator and simple pack indicator. When any of these values change during the course of processing (i.e., the current value is different than that of the previous record), then a transfer will be created, taking total quantities and availability into consideration (see replenish_item(), below).

Program Flow

N/A

Shared Modules

GET_REPL_ORDER_QTY_SQL.REPL_METHOD: Stored PL/SQL procedure for calculating the ROQ of an item at a location.

REPLENISHMENT_SQL.GET_STORE_REVIEW_TIME: Stored PL/SQL procedure for calculating the time between scheduled shipments to a store from a warehouse. This time is used by GET_REPL_ORDER_QTY_SQL in its calculations.

ITEMLOC_QUANTITY_SQL.GET_WH_CURRENT_AVAIL: Stored PL/SQL procedure for calculating the amount of a given item available at a given warehouse.

NEXT_TRANSFER_NUMBER: Stored PL/SQL procedure used for getting the next valid transfer number for use in creating new transfers.

RMS_ROUND_TO_PACKSIZE: Shared C function (see rpl.h) used in rounding an item's quantity up to the size of a simple pack, or for rounding an order quantity up to a receivable pack size.

Data Structures

repl_info_struct: Holds information fetched from the driving cursor.

store_struct: Holds information about item-location combinations, used for ROQ and distribution calculations.

alt_item_struct: Holds information about alternate items associated with a given master item. Used in distribution calculations.

tsfhead_struct: Used to buffer inserts to the TSFHEAD table.

tsfdetail_struct: Used to buffer inserts and updates to the TSFDETAIL table.

item_loc_struct: Used to buffer updates to the item-location tables (RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH, PACKWH).

repl_results_struct: Used to buffer inserts to the REPL_RESULTS table.

domain_struct: Used to cache forecasting domain information.

Function Level Description

General Controlling Functions

main()

The standard Retek main function, this calls init(), process() and final(), and posts messages to the daily log files.

init()

Initializes the Restart-Recovery API and fetches system-level global variables.

driving_cursor()

Opens, fetches data from, or closes the driving cursor. This is a support function for process().

process()

This function fetches records from the driving cursor (driving_cursor()), passes them to replenish_item() to perform all appropriate actions, and commits work when appropriate (post_all(), restart_commit()).

replenish_item()

The controlling function for replenishment calculations. This function copies records out of the driving cursor buffer (copy_repl_to_store()), and calculates the ROQ for each record (get_repl_order_qty()). If a change in item, source warehouse, order control indicator, or simple pack indicator has occurred, the appropriate functions are called to calculate distribution of need over all appropriate alternate items and stores, and to place the transfers. If item's ROQ is zero or negative, no mater simple pack indicator is on the master item will be used for replenishment (build_pack_ratio(), calc_pack_dist(), calc_sub_dist()).

place_tsf_line_item()

This function takes a item-location combination and a transfer quantity, and actually builds the transfer line item (handle_tsf()). It then updates the item-location tables to reflect the change in stock (handle_item_loc()), and writes a record to the reporting table (handle_repl_results()) when appropriate.

final()

The standard Retek final function, this closes down the Restart-Recovery API.

Simple Pack Distribution and Transfer

build_pack_ratio()

Calculates distribution of the master item's recommended order quantity (ROQ) over simple packs. Simple packs are sellable and orderable packs containing only a single item (e.g., six-pack of cola). Since the cost per item will always be less in a pack than singularly, the item will only be ordered in terms of simple packs (if any are applicable). This function tries to divide the total ROQ for the item among all applicable simple packs by using the packs' relative sales history to build a distribution 'mask' containing ratios used to calculate each pack's share of the ROQ. This mask is then adjusted to account for availability (shortages of some packs, surpluses of others).

This function performs the following steps to optimally distribute the ROQ among any and all simple packs:

- If a primary simple pack was defined for this item, that pack will be the only one used to supply the item (`add_primary_pack()`).
- If no primary pack was defined, the program will build a list of all simple packs associated with the item (`get_multi_simple_pack()`).
- If no appropriate simple packs are found, the item will be ordered as itself (`add_single_item()`).
- The historical sales for all simple packs and the master item are added up.
- The ROQ is distributed among the simple packs by taking the ratio of each pack's historical sales to the total historical sales (`first_ratio_pass()`).
- If the first pass through the list did not account for the entire ROQ because of lack of availability for some packs, the program must keep cycling through the items until it has either distributed the ROQ among all available packs or there is simply no available stock left to supply the need (`next_ratio_pass()`).

`first_ratio_pass()`

Performs initial distribution of an item's ROQ among its associated simple packs. Calculates each pack's share as a ratio of its historical sales to the total historical sales (`adjust_pack_ratio()`). The historical sales of the master item are added to those of the simple pack with the lowest cost to give it a greater share of the ROQ. This is a support function for `build_pack_ratio()`.

`next_ratio_pass()`

This function readjusts the ratios of still-available packs to try and cover the share of ROQ not yet allocated, still distributing the leftover ROQ proportionally by historical sales. This is a support function for `build_pack_ratio()`.

`adjust_pack_ratio()`

Sets a simple pack's share of the ROQ to reflect its desirability (in terms of historical sales patterns), adjusting for availability. This is a support function for `first_ratio_pass()` and `next_ratio_pass()`.

`add_primary_pack()`

If an item is flagged to have a primary simple pack, that pack is the only one that will be transferred. This function adds the primary pack to the simple pack distribution array and assigns it the full share of the ROQ. This is a support function for `build_pack_ratio()`.

`get_multi_simple_pack()`

Finds all simple packs associated with a given master item and information about them (historical sales, availability, etc). This is a support function for `build_pack_ratio()`.

`get_single_sales_hist()`

Gets the historical sales of the master item at all stores supplied by the given warehouse for use in calculating distribution among simple packs. Since the master item will only be transferred as a pack, this sales amount will be added to that of the pack with the lowest cost, increasing its share of the ROQ. This is a support function for `build_pack_ratio()`.

`add_single_item()`

If an item is flagged to use simple packs, but none are found, it will be ordered as itself. This function adds the master item to the simple pack distribution structure and assigns it the full share of the ROQ. This is a support function for `build_pack_ratio()`.

`calc_pack_dist()`

Once each simple pack's share of the item's ROQ has been calculated in `build_pack_ratio()`, this function calculates actual transfer quantities and places the transfer line items (`place_tsf_line_item()`). The function loops through each pack in the list, calculating the amount of the pack to transfer to each store (`calc_pack_tsf_qty()`). If the total transfer quantity of the pack exceeds its availability at the warehouse, each store will have its quantity reduced by one receivable pack until a reasonable number has been reached. Finally, a transfer line item is placed for the pack to the store.

`calc_pack_tsf_qty()`

Calculate the actual quantity to transfer for a store based on an alternate item's share of the ROQ at a store, adjusted for any applicable simple pack and/or shipping pack sizes. This is a support function for `calc_pack_dist()`.

Substitute Item Distribution and Transfer

`calc_sub_dist()`

Calculate distribution of the ROQ over substitute items. Substitute items are items (selected by the user beforehand) that can be requested in place of a given item to cover situations where availability is too low or demand is too high.

After calling `get_sub_items()` to generate a list of appropriate items for transfer, the function loops through every item-location combination and performs the following steps to make sure that both need and availability are accounted for when placing transfers from the warehouse to the stores:

- If the total availability of all items in the substitute list cannot cover the full need over all stores, then the ratio of the total availability to the total ROQ is calculated. If total availability *can* cover total ROQ, the ratio is set to 1.
- The initial transfer quantity for the item at the location is calculated as the store's need adjusted by the availability ratio, and rounded up to a receivable pack size.
- If there is not enough of the item available at the warehouse to fill the calculated transfer quantity, the quantity will be decremented to an orderable amount.
- The transfer line item is placed by calling `place_tsf_line_item()`.
- The store's ROQ, total ROQ, availability of the item, and total availability are all decremented by the amount just transferred to prepare for the next item-location's calculation.

`get_sub_items()`

Retrieves substitute items for the current master item and information about them from the database (receiving pack size, availability, etc.). If the fill priority for this set of items (`SUB_ITEMS_HEAD.fill_priority`) is set to 'M'aster, the master item will be the first one in the list, and will be used first to fill need. If it is set to 'S'ubstitute, the master item will be placed at the tail end of the list. This is a support function `calc_sub_dist()`.

`add_master()`

Adds the master item to the appropriate position in the substitutes list. This is a support function for `get_sub_items()`.

shift_subs()

If the fill priority for the substitutes list is set to 'M'aster, the master item must be placed at the head of the list. This function clears out the first position by moving each substitute item 'back' a slot. This is a support function for get_sub_items().

Database DML Handling

post_all()

The DML handling functions (handle_tsf(), handle_item_loc(), handle_repl_results()) normally only post information to the database tables when their respective buffers are full. When a commit point is reached, however, all buffers must be flushed to ensure restartability. This function forces all the buffers to be posted to the database.

handle_tsf()

Controls handling of inserts and updates to the Transfer tables.

add_tsfhead()

Deals with transfer header information. Either finds an appropriate transfer to add line items to (matching to/from locations, department and freight code), or creates a new one. passes back the transfer number for use in add_tsfdetail(). This is a supporting function for handle_tsf().

add_tsfdetail()

Deals with transfer detail information. Either finds an appropriate record on the TSFDETAIL table to add quantity to (matching transfer number and item), or creates a new one if none is found. This is a supporting function for handle_tsf().

get_next_seq_no()

Every line item on a transfer has a unique identifier within that transfer. This function gets the next sequence number for a new line item. This is a supporting function for add_tsfdetail().

post_tsf()

Posts transfer information to the database. Inserts to TSFHEAD, inserts and updates to TSFDETAIL. This is a supporting function for handle_tsf().

handle_item_loc()

Whenever a transfer is created or modified, the source location's reserved quantity and the receiving location's expected quantity must be adjusted to reflect the new stock status. This function controls the handling of updates to the RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH and PACKWH tables.

add_item_loc()

Adds records to arrays for update of expected and reserved quantities on the item-location tables (RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH, PACKWH) based on the appropriate item types. This is a support function for handle_item_loc().

post_item_loc()

Posts item-location stock status changes to the database (RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH, PACKWH). This is a support function for handle_item_loc().

handle_repl_results()

Controls posting of report information to the REPL_RESULTS table.

add_repl_results()

Adds records to the replenishment results structure for reporting. This is a supporting function for handle_repl_results().

post_repl_results()

Posts replenishment information to the REPL_RESULTS table. This is a supporting function for handle_repl_results().

update_review_date()

Updates the last_review_date column on the REPL_ITEM_LOC table to reflect the fact that item-location combinations have just been evaluated.

PL/SQL Stored Procedure Calls

get_repl_order_qty()

Gets the recommended order quantity (ROQ) and other variables for an item-location-supplier and populates elements of a store information structure with them. This function is a wrapper for the REPLENISHMENT_SQL.GET_STORE_REVIEW_TIME and GET_REPL_ORDER_QTY_SQL.REPL_METHOD stored PL/SQL procedures.

`get_wh_current_avail()`

Gets the available quantity of a given item at a given warehouse. This function is a wrapper for the `ITEMLOC_QUANTITY_SQL.GET_WH_CURRENT_AVAIL` stored PL/SQL procedure.

`next_transfer_number()`

Gets the next transfer number in the Oracle stored sequence for creating new transfer headers. This function is a wrapper for the `NEXT_TRANSFER_NUMBER` stored procedure.

Domain Validation

`get_domain_info()`

The `GET_REPL_ORDER_QTY_SQL` stored procedure requires a forecasting domain when performing calculations for item-location combinations using a forecasted method of replenishment (Dynamic or Time Supply). For a given department, class or subclass (depending on the system-level Domain Level option), this function finds the associated forecasting domain. The first time this is called, it will call `build_domain_cache()` to pull all domain information for this thread into an array, and return the desired domain. Subsequent calls will only query the array.

`build_domain_cache()`

Populates a structure with information on department/class/subclass/domain relationships. Because the program is threaded by department, the function will only pull domains associated with departments, classes, or subclass associated with the current thread.

Support Functions

`copy_repl_to_store()`

Copies a record from the structure holding rows from the driving cursor into a structure holding item-location information for ROQ calculation, distribution, and transfer placement.

`reset_store_struct()`

Resets summary variables in a store information structure to prepare it for the next set of line items.

`reset_alt_item_struct()`

Resets summary variables in an alternate item structure to prepare it for the next set of alternates.

Array Sizing

`size_repl_info_struct()`

Allocates memory to the structure used to buffer fetches from the driving cursor.

`size_store_struct()`

Allocates memory to the structure used to hold item-location level information.

`size_alt_item_struct()`

Allocates memory to the structure used to hold information about alternate items (either simple packs or substitute items).

`size_tsfhead_struct()`

Allocates memory to the structure used to buffer inserts to the Transfer Header table.

`size_tsfdetail_struct()`

Allocates memory to structures used to buffer inserts and updates to the Transfer Detail table.

`size_item_loc_struct()`

Allocates memory to structures used to buffer updates of the item-location tables (RAG_SKUS_ST, RAG_SKUS_WH, WIN_STORE, WIN_WH, PACKWH).

`size_repl_results_struct()`

Allocates memory to the structure used to buffer inserts to the Replenishment Results table.

Database Interaction

Tables Selected From:

- DOMAIN_CLASS
- DOMAIN_DEPT
- DOMAIN_SUBCLASS
- ITEM_SUPP_COUNTRY
- PACKHEAD
- PACKITEM
- PACKSTORE_HIST
- PERIOD
- RAG_SKUS_ST_HIST
- REPL_DAY
- REPL_ITEM_LOC
- STORE
- SUB_ITEMS_HEAD
- SUB_ITEMS_DETAIL
- SYSTEM_OPTIONS
- TSFDETAIL
- TSFHEAD
- V_RESTART_DEPT (*View*)
- WH
- WIN_STORE_HIST

Tables Inserted To:

- REPL_RESULTS
- TSFDETAIL
- TSFHEAD

Tables Updated:

- PACKWH
- RAG_SKUS_ST
- RAG_SKUS_WH
- REPL_ITEM_LOC
- TSFDETAIL
- WIN_STORE
- WIN_WH

I/O Specification

N/A

Technical Issues

N/A

RMS to RLS download conversion [rms2rls]

Design Overview

This NAWK program will convert Retek generic download files to the format specified by Nautilus' download file. This involves removing Retek standard flat file specifications, such as FHEAD, FDETL, and FTAIL records, and separating the file by warehouse, so that each specific warehouse will have its own file.

See below for the download programs that produce files to be converted with rms2rls.awk.

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: After processing of download programs, which produce standard Retek flat files, each file needs to be run through the conversion program before Nautilus download programs can run.

Pre-Processing: N/A

Post-Processing: Nawk program will create a .logfile, which will record the success or failure of the conversion process.

Threading Scheme: N/A

Restart Recovery

This is a flat file conversion program done from the Unix shell in an AWK script, no restart/recovery, commit is used or practical.

If there are conversion errors, redo the whole conversion.

Program Flow

N/A

Function Level Description

Read in the first record of the input file (ie. FHEAD)

Assign output file names to variable, OUTFILE, depending upon the file type of the input file located in the FHEAD record.

Process the details of the input file by removing the Retek standard file fields, such as FDETL, THEAD, and TDETL, and writing the record to the correct output file.

Append the DC number to the output file name. Since there may be multiple warehouses processed in one input file, there will be one output file per warehouse.

Write the detail records to the appropriate output file. Some input files will be parsed into several output files, depending upon the Nautilus specified file layout.

Record any errors to the .logfile file.

I/O Specification

Input file: any file produces by the Retek download programs. These files are specified on the command line. The appropriate output file will be produced as specified by the Nautilus download batch programs.

File type	Rms2RIs.awk Output File	Download Program Name
VNND	vendor.dat	supdnld.pc
ITEM	item.dat	itemdnld.pc
BOMD	bill_of_materials.dat	itemdnld.pc
LOCD	ship_dest.dat	locdnld.pc
POHD	po_header.datDC#	Ordrev.pc
PODT	po_detail.datDC#	Ordrev.pc
POHD	po_header.datDC#	Ordprg.pc
TDCH	po_header.datDC#	Tsfdcdld.pc
TDCD	po_detail.datDC#	Tsfdcdld.pc
TDCH	po_header.datDC#	tsfprg.pc
TDCH	po_header.datDC#	tsfalprg.pc
PRHD	rcv_pre_dist_header.datDC#	Ordrev.pc
PRDT	rcv_pre_dist_detail.datDC#	Ordrev.pc
PRDT	rcv_pre_dist_detail.datDC#	Allocupd.pc
CPTT	component_ticketing.datDC#	allocupd.pc
CPTT	component_ticketing.datDC#	Asndnld.pc
CPTT	component_ticketing.datDC#	ordrev.pc
CPTT	component_ticketing.datDC#	tsfdnld.pc
IWOD	inbound_wo.datDC#	powodld.pc

File type	Rms2Rls.awk Output File	Download Program Name
OWOD	outbound_wo.dateDC#	tsfdnld.pc
TSFD	rcv_stock_order_header.datDC#	tsfdnld.pc
TSFD	rcv_stock_order_detail.datDC#	tsfdnld.pc
STOR	rcv_stock_order_header.datDC#	saaldnld.pc
STAL	rcv_stock_order_detail.datDC#	saaldnld.pc
UPCD	upc.dat	itemdnld.pc

Technical Issues

N/A

Maximum Stock Levels for Fashion Items [rpladjf]

Design Overview

Function:

To re-calculate the maximum stock levels for all fashion item locations with a replenishment method of 'F' (floating point). The floating model stock method will dynamically calculate an order-up-to-level. The maximum model stock is calculated using the sales history of various periods of time in order to accommodate seasonality as well as trend. The sales history is obtained from the rag_skus_st_hist/rag_skus_wh_hist tables:

Maximum Model Stock = Sales Rate + service level factor x SQRT(Sales Rate)

Sales Rate = Avg. daily sales for the upcoming 12 weeks of last year (see X in figure 1.0)

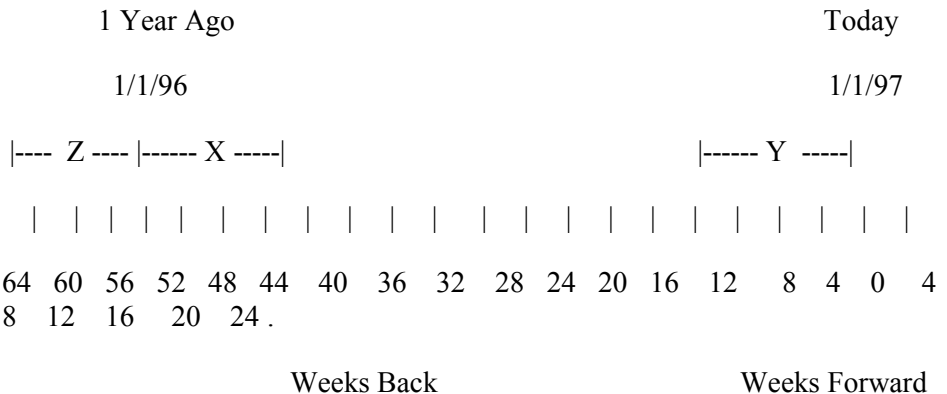
x Avg. daily sales for the previous 12 weeks

(see Y in figure 1.0)

/ Avg. daily sales for the previous 12 weeks of last year

(see Z in figure 1.0)

x (Lead Time + days between reorder cycles)



Service Level Factor = 1 if the user sets the service level to an 85% in-stock service level, or 2 if the user sets the service level to an 98% in-stock service level.

When calculating the sales rates, only the weeks with clean regular sales are taken into account. Weeks with promo or clearance sales are not included. For example, using the following:

Week	1	2	3	4	5	6	7	8	9	10
Sales										
Regular	10	15	3			5	10	0	0	10
Promotion			20	25	20	10				

$$\begin{aligned}\text{Daily Sales Rate} &= (\text{Week 1} + \text{Week 2} + \text{Week 7} + \text{Week 8} + \text{Week 9} \\ &\quad + \text{Week 10})/42 \\ &= (10 + 15 + 10 + 0 + 0 + 10)/42 \\ &= 1.0714\end{aligned}$$

The calculation for the order quantity is the same for this method as for the Constant method (maximum stock - quantity available). The dynamically calculated maximum cannot go below the minimum stock level set by the user.

Substitute items are considered as well. If the item has substitute items and their sales histories are being used (i.e. use sales history indicator set to Yes), the sales histories for the main item and substitute items are summed across the time periods and their totals are used to determine the sales rate ratio.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
REPL_ITEM_LOC	Yes	Yes	No	Yes	No
VSUB_ITEMS_HEAD	No	Yes	No	No	No
SUB_ITEMS_DETAIL	No	Yes	No	No	No
RAG_SKUS_ST_HIST	No	Yes	No	No	No
RAG_SKUS_WH_HIST	No	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	Yes	No	No	No

Scheduling Constraints

Processing Cycle:	PHASE 3
Scheduling Diagram:	Must run before rplextrf/reqextf and after rplatupd.
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	This program is thread by dept.

Restart Recovery

This program implements restart/recovery. The logical unit of work for this program is a valid sku/store combination.

Program Flow

N/A

Shared Modules

REPL_OLT_SQL.GET_OLTS_AND_REVIEW_TIME – to retrieve the number of days until the next review for the given item/location combination.

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Maximum Stock Levels for Staple Items [rpladjs]

Design Overview

Function:

To re-calculate the maximum stock levels for all fashion item locations with a replenishment method of 'F' (floating point). The floating model stock method will dynamically calculate an order-up-to-level. The maximum model stock is calculated using the sales history of various periods of time in order to accommodate seasonality as well as trend. The sales history is obtained from the win_store_hist/win_wh_hist tables:

Maximum Model Stock = Sales Rate + service level factor x SQRT(Sales Rate)

Sales Rate = Avg. daily sales for the upcoming 12 weeks of last year

(see X in figure 1.0)

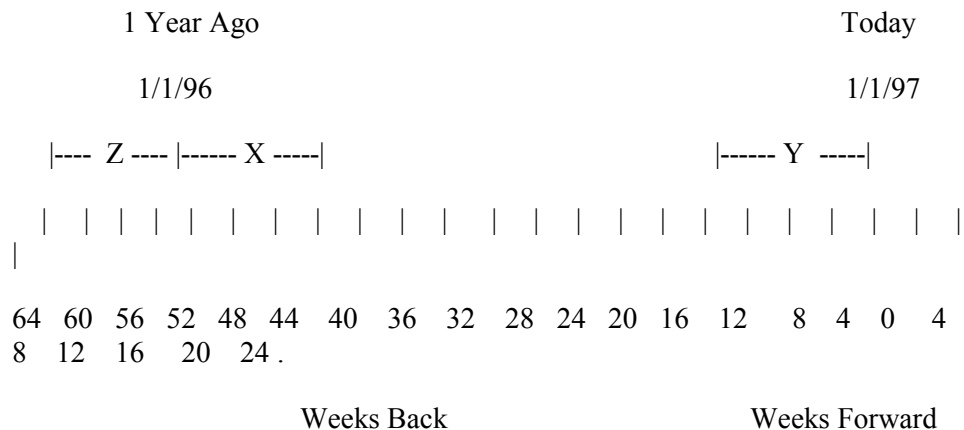
x Avg. daily sales for the previous 12 weeks

(see Y in figure 1.0)

/ Avg. daily sales for the previous 12 weeks of last year

(see Z in figure 1.0)

x (Lead Time + days between reorder cycles)



Service Level Factor = 1 if the user sets the service level to an 85% in-stock service level, or 2 if the user sets the service level to an 98% in-stock service level.

When calculating the sales rates, only the weeks with clean regular sales are taken into account. Weeks with promo or clearance sales are not included. For example, using the following:

Week	1	2	3	4	5	6	7	8	9	10
Sales										
Regular	10	15	3			5	10	0	0	10
Promotion			20	25	20	10				

Daily Sales Rate = (Week 1 + Week 2 + Week 7 + Week 8 + Week 9 + Week 10)/42

$$= (10 + 15 + 10 + 0 + 0 + 10)/42$$

$$= 1.0714$$

The calculation for the order quantity is the same for this method as for the Constant method (maximum stock - quantity available). The dynamically calculated maximum cannot go below the minimum stock level set by the user.

Substitute items are considered as well. If the item has substitute items and their sales histories are being used (i.e. use sales history indicator set to Yes), the sales histories for the main item and substitute items are summed across the time periods and their totals are used to determine the sales rate ratio.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
REPL_ITEM_LOC	Yes	Yes	No	Yes	No
WIN_STORE_HIST	No	Yes	No	No	No
WIN_WH_HIST	No	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	Yes	No	No	No
SUB_ITEMS_HEAD	No	Yes	No	No	No
SUB_ITEMS_DETAIL	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle:	PHASE 3
Scheduling Diagram:	Must run before rplexsts/reqexts and after rplatupd.
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	This program is threaded by dept.

Restart Recovery

This program implements restart/recovery. The Logical unit of work for this function is a valid sku/location combination.

Program Flow

N/A

Shared Modules

REPL_OLT_SQL.GET_OLTS_AND_REVIEW_TIME – Gets the review time and the current and future order leadtimes. It will adjust the leadtimes based on user-specified delivery and the location calendar.

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Replenishment Attribute Update [rplatupd]

Design Overview

Rplatupd.pc (REPL_ATTRIB_UPDATE_DAILY) will execute the logic necessary to maintain replenishment attributes for an item list. (A user can update replenishment attributes for an individual SKU/Style online, but processing an item list would take far too long to be acceptable.) When the Replenishment Attribute Maintenance form (replattr.fmb) is used to maintain the replenishment attributes for an item list, the form calls a package (rplattrb/s.pls) to write the changes to a set of temp tables (REPL_ATTR_UPDATE_ITEM and REPL_ATTR_UPDATE_LOC) listing the items, locations, and attributes to be changed. The batch program will read these tables and process the item location relationships using the information written to the table to determine what replenishment attributes for what locations have to be updated. Items are processed in order of sequence number. Each item list maintenance submittal is given a sequence number.

Rplatupd has been rewritten to comply with changes made to the replenishment data model. Rather than being kept on the four item-location tables, item-location replenishment attributes will now be kept on one table, REPL_ITEM_LOC. In addition, review cycle information will be moved down from the supplier level to the item-location level. This information will now be kept on the REPL_DAY table. These changes allow for easier maintenance of the replenishment attributes across item-locations.

This program is generally only run sporadically, but when it runs, it often has to process very large volumes (e.g., updating the attributes for all SKUs in an entire group or division at all locations – millions of records). The previous version of rplatupd simply fetched records out of the staging tables and called the same PL/SQL package as the online form for each one. While this did keep online performance from being impaired, the program instead took an inordinate amount of time during the nightly batch run, essentially bringing processing to a complete halt for many hours. The problem is not only that updating many records takes longer than updating one, but that the different situations really require not just more time, but significantly different approaches. Therefore, in order to optimize performance, the validation and update logic has been moved out of PL/SQL and into the Pro*C code itself.

TABLE	INDEX	SELECT	INSERT	UPADATE	DELETE
REPL_ATTR_UPDATE_ITEM	Yes	Yes	No	No	No
REPL_ATTR_UPDATE_LOC	Yes	Yes	No	No	No
WIN_STORE	No	Yes	No	Yes	No
WIN_WH	No	Yes	No	Yes	No
RAG_SKUS_ST	No	Yes	No	Yes	No
RAG_SKUS_WH	No	Yes	No	Yes	No
REPL_ITEM_LOC		Yes	Yes	Yes	Yes
REPL_DAY		Yes	Yes	No	Yes
ITEM_SEASONS		Yes	Yes	No	No
SYSTEM_OPTIONS		Yes	No	No	No
ITEM_SUPP_COUNTRY		Yes	No	No	No
RAG_SKUS		Yes	No	No	No
RAG_STYLE		Yes	No	No	No
PACKHEAD		Yes	No	No	No
PACKITEM		Yes	No	No	No
DEPS		Yes	No	No	No

Scheduling Constraints

Processing Cycle: Daily, Phase 3

Scheduling Diagram: This program should run in Phase 3 and be run before the replenishment batch programs, rplads/f, rplex/f, reqexts/f.

Pre-Processing: Truncate the MC_REJECTIONS table (rplatupd_pre)

Post-Processing: Truncate the REPL_ATTR_UPDATE_ITEM and REPL_ATTR_UPDATE_LOC tables (rplatupd_post)

Threading Scheme: Thread by store and warehouse (use v_restart_store_wh view)

Restart Recovery

Driving cursor:

```
EXEC SQL DECLARE c_repl_items CURSOR FOR

/* select all skus (fashion and staple) */

/* union with styles exploded to sku level */

SELECT rau_item.seq_no,

       rau_item.item,

       rau_item.system_ind,

       rau_item.action,

       TO_NUMBER(NULL), /* style */

       rau_item.repl_method_ind,

       rau_item.stock_cat,

       rau_item.repl_order_ctrl,

       rau_item.sourcing_wh,

       TO_CHAR(rau_item.activate_date, 'YYYYMMDD'),

       TO_CHAR(rau_item.deactivate_date, 'YYYYMMDD'),

       rau_item.pres_stock,

       rau_item.demo_stock,

       rau_item.repl_method,

       rau_item.min_stock,

       rau_item.max_stock,

       rau_item.incr_pct,

       rau_item.min_supply_days,

       rau_item.max_supply_days,

       rau_item.time_supply_horizon,

       rau_item.inv_selling_days,
```

```
rau_item.service_level,  
rau_item.lost_sales_factor,  
rau_item.non_scaling_ind,  
rau_item.max_scale_value,  
rau_item.pickup_lead_time,  
rau_item.wh_lead_time,  
rau_item.terminal_stock_qty,  
rau_item.season_id,  
rau_item.phase_id,  
rau_item.supplier,  
rau_item.origin_country_id,  
rau_item.review_cycle,  
rau_item.monday_ind,  
rau_item.tuesday_ind,  
rau_item.wednesday_ind,  
rau_item.thursday_ind,  
rau_item.friday_ind,  
rau_item.saturday_ind,  
rau_item.sunday_ind,  
rau_item.unit_tolerance,  
rau_item.pct_tolerance,  
rau_item.use_tolerance_ind,  
rau_loc.loc,  
rau_loc.loc_type  
FROM v_restart_store_wh vrswh,  
repl_attr_update_loc rau_loc,
```

```

        repl_attr_update_item rau_item
WHERE rau_loc.seq_no    = rau_item.seq_no

AND rau_item.system_ind in ('f','S')

AND vrsw.driver_name = :os_restart_driver_name

AND vrsw.num_threads = :oi_restart_num_threads

AND vrsw.thread_val = :oi_restart_thread_val

AND vrsw.driver_value = rau_loc.loc

AND (rau_item.seq_no > NVL(:os_restart_sequence, -999) OR

      (rau_item.seq_no = :os_restart_sequence AND

        (rau_item.item > :os_restart_item OR

          (rau_item.item = :os_restart_item AND

            (rau_loc.loc > :os_restart_loc))))))

UNION ALL

SELECT rau_item.seq_no,

       rs.sku,

       'f', /* system_ind */

       rau_item.action,

       rs.style,

       rau_item.repl_method_ind,

       rau_item.stock_cat,

       rau_item.repl_order_ctrl,

       rau_item.sourcing_wh,

       TO_CHAR(rau_item.activate_date, 'YYYYMMDD'),

       TO_CHAR(rau_item.deactivate_date, 'YYYYMMDD'),

       rau_item.pres_stock,

       rau_item.demo_stock,

```

rau_item.repl_method,
rau_item.min_stock,
rau_item.max_stock,
rau_item.incr_pct,
rau_item.min_supply_days,
rau_item.max_supply_days,
rau_item.time_supply_horizon,
rau_item.inv_selling_days,
rau_item.service_level,
rau_item.lost_sales_factor,
rau_item.non_scaling_ind,
rau_item.max_scale_value,
rau_item.pickup_lead_time,
rau_item.wh_lead_time,
rau_item.terminal_stock_qty,
rau_item.season_id,
rau_item.phase_id,
rau_item.supplier,
rau_item.origin_country_id,
rau_item.review_cycle,
rau_item.monday_ind,
rau_item.tuesday_ind,
rau_item.wednesday_ind,
rau_item.thursday_ind,
rau_item.friday_ind,
rau_item.saturday_ind,

```

    rau_item.sunday_ind,
    rau_item.unit_tolerance,
    rau_item.pct_tolerance,
    rau_item.use_tolerance_ind,
    rau_loc.loc,
    rau_loc.loc_type
FROM v_restart_store_wh vrswh,
    rag_skus rs,
    repl_attr_update_loc rau_loc,
    repl_attr_update_item rau_item
WHERE rau_loc.seq_no = rau_item.seq_no
AND rs.style = rau_item.item
AND rau_item.system_ind = 'F'
AND vrswh.driver_name = :os_restart_driver_name
AND vrswh.num_threads = :oi_restart_num_threads
AND vrswh.thread_val = :oi_restart_thread_val
AND vrswh.driver_value = rau_loc.loc
AND (rau_item.seq_no > NVL(:os_restart_sequence, -999) OR
    (rau_item.seq_no = :os_restart_sequence AND
    (rau_item.item > :os_restart_item OR
    (rau_item.item = :os_restart_item AND
    (rau_loc.loc > :os_restart_loc))))))
ORDER BY 1,2,44; /* order by seq,item,loc */

```

Shared Modules

ITEMLIST_MC_REJECTS_SQL.INSERT_REJECTS –write rejected records to mc_rejections table for later reporting.

REPL_ATTRIBUTE_MAINTENANCE_SQL.GET_ITEM_TYPE

This package will no longer be called from the program, but the validation logic of these two modules must be kept in sync.

ATTRIBUTE_STRUCT

Will hold an item, location, and all of its replenishment attributes. In addition, it should hold a rowid (to speed up updates). The structure will be used to define the arrays used for both inserts and updates to REPL_ITEM_LOC. It may be desirable to use the same structure type for fetches from the driving cursor, but if that is the case, then a number of extra fields will have to be added to hold the additional columns on REPL_ATTR_UPDATE_ITEM (seq_no, system_ind, action, repl_method_ind, update_repl_days_ind, and the seven update_*day_ind columns). Rather than using a couple dozen scalar variables to hold the old and current attributes when validating, it may make sense to define a second structure similar to this one, but non-arrayed. That way, attributes would be held together within a single envelope and passed more easily

REPL_DAY_STRUCT

This structure will be used for inserts to the REPL_DAY table. It should contain the item, location, location type, and weekday. This array should be sized to seven times the commit max counter, since each item-location combination being fetched could potentially be replenished every day of the week. Because of the need for this larger array, the commit_max_ctr setting on restart_control for this program should be set with caution, since Oracle has a 32k limit on bind arrays. The same structure type can be used to define the arrays for deletes from REPL_DAY, but that array should not hold the weekday, as any delete from this table will clear out all records for an item-location combination.

REPL_ITEM_LOC_DELETE_ARRAY

This array will hold rowids from REPL_ITEM_LOC and will be used to delete records that are being taken off of replenishment.

Function Level Description

init()

initialize restart/recovery

check if contracting is in use

find what day of the week Oracle thinks Sunday is

process()

The controller for most of the processing in the program, this function opens and fetches from the driving cursor according to Retek standards, using array processing. For each item-location coming through (i.e., each record fetched), this function will call the appropriate validation function based on the requested action. If validation passes, information should be passed into the function(s) to fill insert, update, and/or delete arrays. A note on updating attributes: since records on REPL_DAY cannot be updated (weekday is part of the primary key), the old days need to be cleared off and new ones inserted when doing an update. This should only happen, of course, when the user actually wants to change the days of replenishment (update_repl_days_ind = 'Y').

When the fetch array has been fully processed, post functions are called and the restart commit logic is called.

activate_item()

Calls validate_activate. If the item was rejected, call insert_rejection; otherwise call fill_activate_array and fill_insert_days_array to populate arrays.

It also gets the item_season_seq_no for the record's item/season/phase combination.

post_activations

Insert records into repl_item_loc.

update_item

Call validate_update. If the record fails validation, call insert_rejection ; otherwise put it into the update arrays with fill_update_array and fill_insert_days_array.

It also gets the item_season_seq_no for the record's item/season/phase combination.

deactivate_item

Fill the deactivation array, after checking that the record exists on repl_item_loc.

post_deactivations

Delete given rows from repl_item_loc table.

post_day_changes

Insert changes into the repl_day table

get_item_info

Queries the item tables to get the forecast indicator for the item (which will be used during validation for activation and update), as well as the department, class, subclass, and store order multiple. This function also queries DEPS to get the purchase type of the item's department to determine whether or not the item is on consignment or not. If an item is on consignment, it cannot be replenished, since the supplier determines what goes on the shelves. If the item is a fashion SKU and the style was not already retrieved in the driving cursor, this function gets its corresponding style. The function also checks to make sure that the sourcing warehouse is valid.

Get_item_info() also queries ITEM_SUPPLIER and ITEM_SUPP_COUNTRY to verify that the given supplier and origin country (if any) are valid.

Lastly, this function queries PACKHEAD and V_PACKSKU_QTY to get simple pack information. Simple packs are packs containing only a single item, and are indicated by the simple_pack_ind column on PACKHEAD. If the item belongs to any simple packs, the simple_pack_ind on REPL_ITEM_LOC will be set to 'Y'. In addition, if any of the simple packs to which the item belongs is the primary simple pack for the item (indicated by PACKHEAD.primary_simple_pack_ind), the primary_pack_no and primary_pack_qty columns on REPL_ITEM_LOC should be populated with that pack's number and the quantity of the item in that pack, respectively.

Because of the order in which data is processed, this function should only be called when the item changes.

get_itemloc_info

Queries the REPL_ITEM_LOC table to find the current replenishment attributes for the current item-location combination. If no record is found, the item is not on replenishment at that location (this is not necessarily an error). The rowid should also be fetched to facilitate updates and deletes. Queries the appropriate item/store table (RAG_SKUS_ST, WIN_STORE) to get the clearance indicator.

validate_activate

If an item-location is already on replenishment, the function should return a non-fatal error so that the record is skipped. If the item-location meets any of the following criteria, insert_rejection() should be called with the appropriate rejection reason code (see table MC_REJECTION_REASONS), and a non-fatal error should be returned so the record is not written to the activate array:

- The item is not supplied by the specified replenishment supplier/origin country.
- The item is on consignment.
- The replenishment method requires forecast information (Time Supply or Dynamic), but the item is not forecastable.
- It is on clearance and contracting is disabled (this check only applies to stores, since an item cannot be on clearance at a warehouse).
- The item is not stocked at the specified sourcing warehouse (this check only applies to stores, a warehouse is never sourced by another warehouse).
- If a seasonal method of replenishment is being used, this function also calls validate_item_seasons() to ensure that the item-season-phase relationship exists, or creates it if it doesn't.
- If the item-location combination passes all the validation criteria, the function will return successfully.

validate_update

When the user updates replenishment attributes, she may not want to change all of them, and so will leave certain fields blank. These will get written to REPL_ATTR_UPDATE_ITEM as NULL values. Therefore, any column coming out of the driving cursor as NULL is seen as representing a “no-change”. This requires careful monitoring of indicator variables during validation and assignment.

If an item-location is not already on replenishment, the function should return a non-fatal error so that the record is skipped. If the item-location meets any of the following criteria, insert_rejection() should be called with the appropriate rejection reason code (see table MC_REJECTION_REASONS), and a non-fatal error should be returned so the record is not written to the update array:

- The item is not supplied by the specified replenishment supplier/origin country.
- The replenishment method requires forecast information (Time Supply or Dynamic), but the item is not forecastable.

- The item is not stocked at the specified sourcing warehouse (this check only applies to stores, a warehouse is never sourced by another warehouse).
- If the activation date is being changed but the deactivation date is not, and the new activation date is later than the old deactivation date.
- If the deactivation date is being changed but the activation date is not, and the new deactivation date is earlier than the old activation date.
- If the minimum stock quantity is being changed but the maximum is not, and the new minimum is greater than the old maximum.
- If the maximum stock quantity is being changed but the minimum is not, and the new maximum is less than the old minimum.
- If the minimum number of supply days is being changed but the maximum is not, and the new minimum is greater than the old maximum.
- If the maximum number of supply days is being changed but the minimum is not, and the new maximum is less than the old minimum.
- If a seasonal method of replenishment is being used, this function also calls `validate_item_seasons()` to ensure that the item-season-phase relationship exists, or creates it if it doesn't.
- If the item-location combination passes all the validation criteria, the function will return successfully.

`fill_activate_array`

Assigns all replenishment attributes for the current record to the array to be used in inserts to `REPL_ITEM_LOC`.

`fill_update_array`

Assigns the replenishment attributes for the current record to the array to be used in updating `REPL_ITEM_LOC`. If the replenishment method is changing (`REPL_ATTR_UPDATE_ITEM.repl_method_ind = 'Y'`), then all replenishment-method-specific attributes must be written as-is to the arrays, whether they are NULL or not. These attributes are: `repl_method`, `min_stock`, `max_stock`, `incr_pct`, `min_supply_days`, `max_supply_days`, `time_supply_horizon`, `inv_selling_days`, `service_level`, `lost_sales_factor`, `terminal_stock_quantity`, `season_id`, and `phase_id`. If the replenishment method is not changing, these attributes should be left unchanged if they are NULL on `REPL_ATTR_UPDATE_ITEM`; that is, the old value from `REPL_ITEM_LOC` should be written to the array rather than the new NULL value to keep the actual record on the table from being inadvertently overwritten. All other attributes should be left unchanged if they are NULL on `REPL_ATTR_UPDATE_ITEM`.

fill_insert_days_array

Assigns the item, location, location type, and weekday to the array to be used in inserting to REPL_DAY. A record needs to be added for each weekday that has been marked as a review day for the current item-location combination.

fill_delete_days_array

Assigns the item and location to the array to be used in deleting from REPL_DAY. The weekday does not need to be added to this array, since the program will simply delete all weekdays for a given item-location combination.

open_dc

Open the driving cursor.

fetch_dc

Fetch the driving cursor.

post_updates

Performs arrayed update of REPL_ITEM_LOC.

validate_item_seasons

If a Seasonal method of replenishment (either Dynamic or Time-Supply) is being used, a record on the ITEM_SEASONS table must exist to maintain the foreign key dependency from REPL_ITEM_LOC. This function queries ITEM_SEASONS to ensure that a relationship exists. If it doesn't, a record should be inserted. No error should be raised on a DUP_VAL_FOUND occurrence, since it is possible that another thread of the program could be trying to write the same record, as threading is done by location rather than item. It is not possible to perform this insert as an arrayed SQL statement, since a DUP_VAL_FOUND error would stop the rest of the array from being posted.

insert_rejection

This function calls the stored PL/SQL procedure ITEM_LIST_MC_REJECTS_SQL.INSERT_REJECTS, passing in the error information passed into this function by the caller. It should take in a rejection reason and up to three other strings (for additional error message text) as parameters.

size_arrays

Allocates memory for all processing arrays.

final()

call restart_close to finish restart/recovery

Technical Issues

N/A

Replenishment Order Build [rplbld]

Design Overview

Rplbld builds RMS orders from Recommended Order Quantities (ROQ) generated by rplext.pc. Cntrprss.pc associates contracts with the ROQ created by rplext.pc. These ROQs are placed on a temporary table (ORD_TEMP) by rplext.pc. All records on ORD_TEMP are processed by rplbld each night. These ORD_TEMP records are placed into logical groups, and a RMS order is created for each logical group.

In order to be placed in the same order group, the item/location ROQs from ORD_TEMP must share a common supplier, have the same order_status ('W'orksheet or 'A'pproved), and be on the same contract (or not be associated with a contract). Depending on flags on the ORD_INV_MGMT table, two other criteria can be used for splitting order groups. First, if the INV_MGMT_LVL is set to 'D'ept, only items in a single department are allowed in an ordering group. Second, the SINGLE_LOC_IND can be set to 'Y'es. If this is the case, only one location is allowed per ordering group. Finally, sku may only exist in an ordering group with a single origin country. When an item/loc ROQ ORD_TEMP record is encountered with a different origin country than the one it exists with in the current ordering group, it is placed in a different ordering group.

To assist the recalculation and order scaling processes, the REPL_RESULTS record, associated with the ORD_TEMP being processed, is updated with the ORDER_NO and ALLOC_NO that the ORD_TEMP record was placed with.

TABLE	SELECT	INSERT	UPDATE	DELETE
ORD_TEMP	Yes	No	No	No
REPL_RESULTS	Yes	No	Yes	No
ORDHEAD	No	YES	No	No
ORDSKU	No	YES	No	No
ORDLOC	No	YES	No	No
ALLOC_HEADER	No	YES	No	No
ALLOC_DETAIL	No	YES	No	No
CONTRACT_HEADER	YES	No	Yes	No
CONTRACT_DETAIL	No	No	Yes	No
SUP_INV_MGMT	YES	No	No	No
ORD_INV_MGMT	No	Yes	No	No
SUPS	Yes	No	No	No
ADDR	Yes	No	No	No

TABLE	SELECT	INSERT	UPDATE	DELETE
SUP_IMPORT_ATTR	Yes	No	No	No
DEPS	Yes	No	No	No
UPC_EAN	Yes	No	No	No
UPC_SUPPLIER	Yes	No	No	No
WIN_STORE	Yes	No	No	No
WIN_WH	Yes	No	No	No
RAG_SKUS_ST	Yes	No	No	No
RAG_SKUS_WH	Yes	No	No	No
PACKSTORE	Yes	No	No	No
PACKWH	Yes	No	No	No
SYSTEM_OPTIONS	Yes	No	No	No
PERIOD	Yes	No	No	No

Scheduling Constraints

Processing Cycle Phase III

Pre-Processing: rplext.pc, cntprss.pc(if contracting is being used),
prepost.pc (prepost user/password rplbld pre)

Post-Processing: supcnstr.pc

Threading Scheme: SUPPLIER

Restart Recovery

The logical unit of work is supplier, contract, and order status. The driving cursor is ordered by supplier, contract, order status, sku, location, crossdock indicator, and crossdock store. The first three elements are for restart recovery and order splitting. The next four are for performance and to ensure crossdock posting performs correctly. When the supplier, contract, or order status changes, there is an opportunity to commit. At that point the number of orders created since the last commit is compared with the commit max counter. If the number of orders is greater than the commit max counter, retek_force_commit is called.

Program Flow

N/A

Shared Modules

ORDER_NUMBER_SQL.NEXT_ORDER_NUMBER – Uses a database sequence to generate an order number to assign to each order created by the program.

CAL_TO_454_LDOW – Used to setup dates for the ORDHEAD insert.

ORDER_SETUP_SQL.DEFAULT_ORDHEAD_DOCS – Defaults required documents needed for each order created by the program.

CURRENCY_SQL.GET_RATE – Retrieves the currency rate for the supplier that the program is creating an order for.

ORDER_EXPENSE_SQL.INSERT_COST_COMP – Inserts needed cost components for an order flagged for ELC.

ORDER HTS_SQL.DEFAULT_CALC HTS – Inserts needed HTS information for import orders.

ORDER_SETUP_SQL.DEFAULT_ORDSKU_DOCS – Defaults the required documents for each sku orders created by the program.

NEXT_ALLOC_NO - Uses a database sequence to generate an allocation number to assign to each allocation created by the program.

NEW_PACK_LOC – Creates a pack location relationship in RMS. Rplex.pc does not verify a pack exists at a location before ordering it to that locaiton.

PRICING_ATTRIB_SQL.BUILD_PACK_RETAIL – Retrieves the retail price for a pack item given a location.

Data Structures

Driving cursor fetch structure

This structure is only used to exchange information brought back by the driving cursor with the rest of the program. It is also the only structure that contains arrays. The rest of the DML statements in the program do not use array processing.

Process structure

This structure holds all of the information as the driving cursor fetch array plus additional housekeeping and contraction information. This structure is used to move driving cursor information between the driving cursor and the linked lists

ordloc linked list

Each node of this linked list holds the information needed to insert an ORDLOC or ALLOC_DETAIL (and potentially ALLOC_HEADER) record. It also is used to update the REPL_RESULTS table.

Ordsku linked list

Each node of this linked list holds the information needed to insert an ORDSKU record. Each node also holds a ordloc linked list pointer.

Ordhead linked list

Each node of this list holds the information needed to insert a ORDHEAD record and some bookkeeping info. It also hold a supplier info structure and a pointer to a ordsku linked list.

Supplier info structure

This structure holds the needed information from the SUPS, SUP_IMPORT_ATTR, ADDR, DEPS, and SUP_INV_MGMT tables for each ordhead node.

Function Level Descriptions

main()

The standard Retek main function, this calls init(), process(), and posts to the daily log files.

init()

This function initializes the restart/recovery API and fetches global options and variables.

Group 1 - setup support functions

These functions get records from the driving cursor and gather additional information to get them ready to pass to the second group of functions, the linked list building functions.

Process()

This is the main control function of the program. It contains the driving cursor. The driving cursor is fetched using array processing, it is fetched in groups of the commit max counter. The records brought back by the driving cursor are looped through individually. For each record, the `get_sup_info()` and `get_contract_info()` functions are fetched to fill out additional information that can not be supplied by the driving cursor. Once all the need information is gathered, a decision is made whether to add the current record to the linked lists or to process the linked lists and start over. This decision is made by calling the `switching_variable_change()` function. This function will return one when the conditions dictate a flush of the linked lists. The flushing triggers are a change is supplier, a change is order status or a change is contract number. If the verdict is a flush of the linked lists, the `do_inserts()` function is called to do just that. It also clears out the linked lists and frees their memory. For each order created a counter is incremented. When this counter becomes greater than the commit max counter, a commit is done by calling `retek_force_commit()`. As the logical unit or work is supplier, order_status, and contract_no, The commit can only be done when one of these changes(note they are the same as the flushing triggers). Finely, `add_to_order()` is called to add the current record to the linked lists. If the linked lists were just flushed, or the first record is being processed, the linked lists could be empty.

Size_driv_cur_struct()

Allocates memory for the structure used in the array fetch of the driving cursor. The structure is sized to the commit max counter.

Copy_driv_cur_rec()

This functions copies a single record of the driving cursor fetch structure to a single element holder structure (process structure). The single element structure it then used to gather additional needed information, and is passed to the linked list building functions. The single element structure allow the rest of the program to not use any array subscripts pertaining to the driving cursor for loop.

Switching_variables_change()

This function determines if the supplier, contract, or order status has changed from the last call to it. If no change has occurred, zero is returned. If a change has taken place one is returned.

Get_sup_info()

This function gathers information from SUPS, SUP_IMPORT_ATTR, ADDR, DEPS, AND SUP_INV_MGMT for each record fetched from the driving cursor. It uses static local variables to avoid looking at these tables any more than necessary. The logic here needs to deal with the fact that information can be stored on SUP_INV_MGMT at the supplier or the supplier/department levels. Which level it is stored at, is determined by SUPS.INV_MGMT_LVL. When this is 'D'epartment, SUP_INV_MGMT is searched at the supplier/department level. If no record is found, SUP_INV_MGMT is searched at the supplier level. If a record is still not found, default values are used. When the SUPS.INV_MGMT_LVL is 'S'upplier, SUP_INV_MGMT is searched at the supplier level. If no record is found, default values are used. Once, the values are found(from the tables or the defaults are used), fill_sup_struct() is called.

Fill_sup_struct()

This function populates a supplier info structure with passed in information gathered in get_sup_info(). The supplier info struct is used through out the rest of the program. Each order head node in the linked lists is associated with a supplier info structure.

Get_contract_info()

This function retrieves contract information for the passed in process structure. It uses local static variables to avoid unnecessary database hits. If a database hit is required, the contract terms, approval indicator, and currency code are brought from CONTRACT_HEADER for the passed in contract number (in the process structure).

Group 2 - linked list building functions

These functions take in a process structure and add it to the ordhead, ordsku, and ordloc linked lists.

Add_to_order()

This function takes in a process structure and adds it to the current ordering linked lists. If the record cannot be added to any existing orders, or the lists are empty, this function will create a new node in the ordhead head list.

It loops on the ordhead list and tries to add the passed in record to each order by calling the add_sku() function. Certain conditions need to be met before the add_sku function. These conditions deal with order splitting variables. Replenishment can be set up to split orders by dept. If this flag is on, the dept of the passed in record must match the dept that is stored at the ordhead node before add_sku() is called. Orders can also be limited to only allow one location per order. If this flag is set, the location of the sent in record must match the location stored at the ordhead node level.

Each order is also assigned a write indicator. This indicator determines if the order will actually be created by the order posting functions. The rules for setting this variable are below:

- If due_processing is turned off for the current vendor line, the ordhead write flag is turned on as soon as one of the item/locs is found with a 'Y' due_ind.
- If due processing is turned on and the non_due_create_flag is 'Y', the write flag is always 'Y'
- If due processing is turned on and the non_due_create_flag is 'N' and the due level is 'L', the write flag is turned on as soon as one item/location is added that has an expected stock out (ESO) greater than its accepted stock out (SSO).
- If due processing is turned on and the non_due_create_flag is 'N' and the due level is 'O', the write flag is defaulted to 'Y'. It cannot be determined if the order will be written until all the item/locs are added, so logic is in insert_ordhead to make this decision in that case.

Other bookkeeping is also done at the ordhead level. ESO and ASO are summed, lead times are tracked, the import indicator is set.

Add_sku()

This function takes in a process structure and a sku linked list. If the passed in sku is already in the list with the same origin country, add_loc() is called to add the location to the location list associated with the sku. If the sku is in the list with a different origin country, control is returned to add_to_order() with a flag set telling it to try the next order(a sku can only exist on a order with one origin country). If the sku is not in the sku list, it is added and add_loc is called().

This function tracks whether or not the sku needs to be placed on the order when the order is created. This logic is below:

- If due processing is on, the sku is marked for addition to the order when its for one location qty is > than zero.
- If due processing is off, the sku is marked for addition to the order when the item/loc due flag is set to 'Y'es.

Add_loc()

This function takes in a process structure and a location linked list. If the location is not already in the linked list it is added. If the location is in the list, its qty is incremented. A write flag is also tracked at the ORDLOC level. The logic around it is below.

- If due processing is on, the loc is marked for addition to the order when the qty is > than zero.
- If due processing is off, and the record is a cross dock total record, the location is marked for addition to the order when the qty is > than zero.
- If due processing is on, and the record is not a cross dock total record, the location is marked for addition to the order when the item/loc due flag is set to 'Y'es.

Group 3 - order posting functions

Functions in this group take the linked list built by the functions in group 2 and create RMS orders for the information in them.

Do_inserts()

This functions loops through the linked lists and calls the necessary insert functions. There are 3 nested while loops. The first loop steps through all the nodes in the ordhead list. If the write flag is set, an insert_ordhead() is called, else it moves to the next order. The middle loop steps through the ordsku list associated with the current ordhead node. For each node in the ordsku list insert_ordsku() is called. The inner most loop steps through the ordloc list associated with the current ordsku node that is associated with the current ordhead node. For each node calls are made to the insert_ordloc() and insert_alloc_header(), insert_alloc_detail(), or just insert_ordloc depending on what type of record is contained in the ordloc node.

The following shows a different representation of the above paragraph.

LOOP ON ORDHEAD

insert ordhead record

insert ord_inv_mgmt record

call necessary packages for new order creation

LOOP ON ORDSKU in the current ORDHEAD node

insert ordsku record

call necessary package for new ordsku creation

LOOP ON ORDLOC in the current ORDSKU node in the current ORDHEAD node

if wh portion of xdock

-- These will always come before the store portions of the xdock ensuring a allocation header will already be created when the store portions are reached because of the ordering of the driving cursor.

insert a ordloc record (total xdock record)

create an allocation header

update repl_results with the order number

else if store portion of xdock

-- the allocation header record will always be there already due to the ordering of the driving cursor

insert an allocation detail record

update repl_results with the order and allocation numbers

else direct to store or direct to wh

update repl_results with the order number

end if

END ORDLOC LOOP

If elc is on, call add_cost_comp(), (the package needs the ordloc records to be there to work)

END ORDSKU LOOP

END ORDHEAD LOOP

insert_ordhead()

This function takes in a ordhead node pointer and inserts a record in to ORDHEAD. First check if the order should be created at all, if it should not, return a one. Then it calls build_dates() and get_exchange() to set up some info for the insert. Next the actual record is inserted into ORDHEAD using the information in the passed in ordhead node. After the record is inserted, header_defaults(), update_contract(), and insert_ord_inv_mgmt() are called to clean up the ORDHEAD creation.

Insert_ord_inv_mgmt()

This function takes in a SUP_INV_MGMT rowid(populated by get_sup_info()) and inserts a row into ORD_INV_MGMT by doing an insert select with the passed in rowid. If no rowid is available, a default record is inserted.

build_dates()

This function takes in the min and max lead time from an ordhead node and populates a not before date, not after date, and an end of week date for the insert into ORDHEAD.

Get_exchange_rate()

This function is a wrapper for the CURRENCY_SQL.GET_RATE. The package is called sending in the currency code of the order, it returns the exchange rate that is used in the insert to ORDHEAD.

Update_contract()

This function will update the contracting tables if the order being inserted is a contracting order. If the contract being used is a type A contract, CONTRACT_DETAIL is updated with the new qty, else CONTRACT_HEADER is updated with the new cost (using the rowid fetched in get_contract_info)).

Insert_ordsku()

This function inserts a record into ORDSKU. First, UPC info is retrieved by calling get_upc_info(). Next, the record is inserted into ORDSKU.. Finally, sku_defaults() is called to clean up the ORDSKU insert.

Get_upc_info()

Given a sku and supplier, this function gets the upc and upc_supplement from UPC_EAN and UPC_SUPPLIER for the insert into ORDSKU.

Add_comp_cost()

This function is a wrapper for a call to the ORDER_EXPENSE_SQL.INSERT_COST_COMP and ORDER HTS_SQL.DEFAULT_CALC HTS package calls. It is called for each record inserted into ORDSKU when the elc indicator is on to default/add expense and HTS components to the order/item to achieve an estimated landed cost for the order/item.

Sku_defaults()

This a wrapper for a call to the ORDER_SETUP_SQL.DEFAULT_ORDSKU_DOCS package. It is called for each record inserted into ORDSKU.

Insert_alloc_header()

Given a ordloc node pointer, this function will insert a record into ALLOC_HEADER. The record will only be created when the write flag is set to Y on the ordloc node.

Get_alloc_no()

This function will populate a variable with a fresh allocation number by calling the NEXT_ALLOC_NO procedure. This function is called for each allocation that is written(called each time insert_alloc_header() is called).

Insert_ordloc()

This function inserts a record into ORDLOC. For each ordloc node sent to this function, the associated REPL_RESULTS, if any, record is updated by calling update_repl_results(). After the REPL_RESULTS record is updated, a check is done to see if the ORDLOC record will actually be created. If so, the unit retail to be used in the insert is retrieved by calling get_unit_retail(). Then, the ORDLOC record for the passed in ordloc node is inserted.

get_unit_retail()

Retrieves the unit retail for the passed in item location for each record inserted into ORDLOC. This is done for staple and fashion skus by using cursors against the item/location tables. If the item is a pack, the create_pack_loc() and build_pack_retail functions are used.

Create_pack_loc()

This function is a wrapper for a call to the PL/SQL function NEW_PACK_LOC. It first checks if the pack/location relationship exists, if it doesn't, the function is called. This is necessitated by the fact that rplext.pc will create records on ORD_TEMP without bothering to check if the pack relationship exists. If this relationship doesn't exist, build_pack_retail would fail.

Build_pack_retail()

This is a wrapper for a call to PRICING_ATTRIB_SQL.BUILD_PACK_RETAIL.

Insert_alloc_detail()

This function will insert a record into ALLOC_DETAIL. Update_repl_results() is always called. The insert into ALLOC_DETAIL is only done when the write flag associated with the passed in ordloc node is 'Y'es. If the item associated with the ordloc node is a pack, create_pack_loc() is called.

Update_repl_results()

This function updates the REPL_RESULTS with the passed in order_no and alloc_no. It does this update using the passed in rowid(from the driving cursor). If the rowid is null(xdock total record), it simply returns control to the calling function.

I/O Specification

N/A

Technical Issues

N/A

Vendor Replenishment Extraction [rplex]

Design Overview

Rplex (Vendor Replenishment Extraction) is the driving program for the replenishment process. It cycles through every item-location combination that is set to be reviewed on the current day, and calculates the quantity of the item that needs to be ordered to the location (if any). In addition, it distributes this Recommended Order Quantity (ROQ) over any applicable simple packs (sellable and orderable packs that contain only a single item – for instance, a six-pack of cola or a twelve-pack of socks). The program then writes these order line items to a temp table (ORD_TEMP) for processing by cntrprss and rplbld.

The logic of this program is determined mainly by the stock category of the item-location combination. If the stock category is Direct to Store, the process is fairly straightforward. The ROQ is calculated, and this quantity is then distributed among any applicable simple packs.

If it is Crossdocked, the ROQ and distribution calculations are handled similarly to the Direct to Store category. However, when crossdocking the item must be ordered to a warehouse before being allocated to the store. This means that the total number of items being ordered must be tracked across all stores sharing the same source warehouse, so that a summary record can be written to ORD_TEMP to connect all the stores and create one order to the warehouse to be allocated from there.

No item-store combinations with a stock category of Warehouse Stocked are processed by this program. Since these stores are supplied via transfers from the source warehouse rather than purchase orders, they are processed in a separate program (reqext). However, orders must still be built so that the source warehouse has stock to supply the stores with. If the replenishment method for the item-warehouse combination is non-forecasted (Constant, Floating-Point, or Min-Max), the process is the same as Direct to Store. However, if the method does involve forecasting (Dynamic or Time-Supply), then the ROQ for the warehouse is based on the predicted ROQ (or need) of the item (and its associated simple packs) at all stores that are supplied by the warehouse.

In all cases, the ROQ must be distributed over an item's associated simple packs (if any). If the item does not use simple packs, then the issue is moot – the item will be ordered on its own. If there is only one pack associated with the item (referred to as the primary simple pack), then there is no distribution needed – the item will be ordered in this simple pack, since the cost per item for a pack is always less than that of an individual item. If multiple simple packs can be substituted for an item, then the distribution of the ROQ over these packs is determined by comparing the packs' relative sales history. Replenishing an item through multiple simple packs can have a severely negative effect on the performance of this program! Because the pack distribution depends on access to the huge sales history tables (PACKSTORE_HIST, RAG_SKUS_ST_HIST, WIN_STORE_HIST), it is not recommended that many items be placed on replenishment through multiple simple packs. Whenever possible, it is better to assign a primary simple pack to the item, since this does not require distribution calculation.

Once the order quantity for an item (or simple pack) has been calculated, an order line item is written to the ORD_TEMP table if 1) the actual quantity to order is greater than zero, and 2) the replenishment order control indicator for the item-location combination is either Automatic or Semi-Automatic. If it is Manual, a record will be written to another table (REPL_RESULTS) for reporting purposes. If the system-level Replenishment Results indicator is set to "Yes", then all order line items being written to ORD_TEMP will also be written to REPL_RESULTS. If the system-level All Replenishment Results indicator is set to "Yes", all line items will be written to REPL_RESULTS, even if the quantity to order is zero. Crossdock summary records are never written to REPL_RESULTS, since they are only used internally within the replenishment batch process.

Accepted_stock_out (ASO) This value will be distributed over multiple simple packs. When multi-simple packs are encountered, a ratio is assigned to each simple pack based on their sales history. The main item's ASO (populated by the ROQ PKG) will be split over the multi-simple packs by multiplying it by this ratio for each of the simple packs. This will be done in build_pack_dist_struct(). In the for loop for simple packs, before the call to calc_pack_order_qty, set the aso for the pack_dist_struct. Set it equal to the repl_info_struct's aso * the packs ratio. After the for loop set the last item's aso the same way, then add the single items ratio of the aso to it.

Expected_stock_out (ESO) This value will be distributed over multiple simple packs. When multi-simple packs are encountered, a ratio is assigned to each simple pack based on their sales history. The main item's ESO (populated by the ROQ PKG) will be split over the multi-simple packs by multiplying it by this ratio for each of the simple packs. This will be done in build_pack_dist_struct(). In the for loop for simple packs, before the call to calc_pack_order_qty, set the eso for the pack_dist_struct. Set it equal to the repl_info_struct's eso * the packs ratio. After the for loop set the last item's eso the same way, then add the single items ratio of the eso to it.

Scheduling Constraints

Processing Cycle Phase III

Scheduling Diagram: Pccext, rplatupd, rpladjf, rpladj, rext and cntordb need to run before rplext to make sure that replenishment attributes and stock information are all up to date. If contracting is being used, cntrprss should run after rplext; otherwise, run rplbld.

Pre-Processing: The ORD_TEMP table is truncated in the pre-processing.

Post-Processing: N/A

Threading Scheme: DEPT

Restart Recovery

The logical unit of work is item, supplier. The driving cursor is ordered by item, supplier, sourcing warehouse, and order control indicator. The first two elements are for restart recovery. The second two have to do with crossdocked orders. Because a summary record has to be written for a crossdock, totals must be kept across all locations an item is being ordered to. When the sourcing warehouse changes, we know we have reached the end of this particular crossdock and can post the summary record. The order control indicator is needed since if some crossdocks are written to be Semi-automatically approved (written in Worksheet status) and others to be Automatically approved, two separate orders (and thus two separate summaries) need to be written.

Program Flow

N/A

Shared Modules

GET_REPL_ORDER_QTY_SQL.REPL_METHOD: Stored PL/SQL procedure for calculating the ROQ of an item at a location.

REPLENISHMENT_SQL.GET_ITEM_LOC_REVIEW_TIME: Stored PL/SQL procedure for determining the time between reviews for an item-location combination. This information is used in GET_REPL_ORDER_QTY and is posted to REPL_RESULTS.

RMS_ROUND_TO_PACKSIZE: Shared C function (see rpl.h) used in rounding an item's quantity up to the size of a simple pack, or for rounding an order quantity up to a receivable pack size.

Data Structures

`repl_info_struct`: Holds information fetched in from the driving cursor, as well as storing variables to be used in other calculations.

`ord_temp_struct`: Used to buffer inserts to the `ORD_TEMP` table.

`repl_results_struct`: Used to buffer inserts to the `REPL_RESULTS` table.

`pack_dist_struct`: Used for holding information about the simple packs that an item belongs to and their relative contribution to the total order quantity of the item. If an item has no simple packs associated with it, this structure will simply hold information for the singular item itself.

`xdock_info_struct`: Holds information to be used in placing the summary record of a crossdocked order.

`domain_struct`: Used to cache forecasting domain information.

Function Level Descriptions

`main()`

The standard Retek main function, this calls `init()`, `process()`, `final()` and posts to the daily log files.

`init()`

This function initializes the restart/recovery API and fetches global options and variables.

`driving_cursor()`

Opens, fetches or closes the driving cursor. This is a support function for `process()`.

`process()`

Controlling function for processing. Fetches data from the driving cursor (by calling `driving_cursor()`), calls `replenish_item()` to do most of the work for each record returned, posts orders when necessary and handles restart/recovery.

`replenish_item()`

The main controlling function for replenishment calculations.

If the location being processed is a warehouse and the warehouse is using a forecasted method of replenishment (either Dynamic or Time Supply), the calculations for determining ROQ and simple pack distributions are based on the stores that this warehouse is a source for, and `replenish_to_wh()` is called to find this information and place the orders.

Otherwise, this function calls `get_repl_order_qty()` to get the raw ROQ for the item-location combination, then distributes the quantity among any appropriate simple packs (in the `build_pack_dist_struct()` function). If the current record is part of a crossdock order, the totals are updated, and when all the locations for the order have been processed, a summary record is posted. Finally, the order line item is placed by calling `place_item_orders()`.

`build_pack_dist_struct()`

Builds a list of simple packs to order in place of given single item, when applicable. If the item is not associated with any simple packs (determined by the simple pack indicator on the `REPL_ITEM_LOC` table), the item will simply be written to the distribution structure by itself (after rounding the quantity up to an orderable pack size). If a primary simple pack is defined on the `REPL_ITEM_LOC` table, that pack will be written to the distribution structure (again, after rounding).

If multiple simple packs can be used to supply a given item, then the total ROQ must be appropriately divided among these packs. `Get_multi_simple_pack()` is called to retrieve information and sales history for each pack. The following steps are then taken to optimally divide the ROQ among the simple packs:

- The item's total sales over a given number of weeks are calculated by adding up the historical sales for each pack, then adding in the singular item's sales.
- The ratio of individual sales to the total is calculated for each simple pack and the singular item (`calc_pack_dist_ratio()`).
- Each simple pack's actual order quantity is calculated by multiplying the pack's sales ratio by the ROQ, then rounding that up to the appropriate orderable pack size.
- The singular item's share of the ROQ is then added to that of the simple pack with the lowest cost.

`add_single_item()`

Adds a singular item to a simple pack distribution structure. This is a support function for `build_pack_dist_struct()`.

`add_primary_pack()`

Calculates the quantity to order when using a single (primary) simple pack and adds it to the distribution structure. This is a support function for `build_pack_dist_struct()`.

`calc_pack_ratio()`

Calculates the ratio of the historical sales of a simple pack to the overall sales of an item. This is a support function for `build_pack_dist_struct()`.

`calc_pack_order_qty()`

Calculates the actual number of simple packs to order based on the number of singular items in the pack, and the number of packs in a receivable pack. This is a support function for `build_pack_dist_struct()`.

`get_multi_simple_pack()`

If an item can be associated with multiple simple packs, this function will open a cursor to fetch all the applicable packs and their sales history. For each pack, the number ordered will be determined by their relative sales. There are different cursors for stores and warehouses, since the sales history for a warehouse will be the sum of sales histories of all stores supplied by that warehouse. The number of weeks of history to sum over is set at the system level (`SYSTEM_OPTIONS.repl_pack_hist_wks`). This is a support function for `build_pack_dist_struct()`.

`get_sales_hist()`

Gets the historical sales for a singular item. There are different cursors for stores and warehouses, since the sales history for a warehouse will be the sum of sales histories of all stores supplied by that warehouse. This is a support function for `build_pack_dist_struct()`.

`replenish_to_wh()`

Replenishes an item to a warehouse using a forecasted method of replenishment that will in turn be used as a replenishment source for stores (see `reqext`).

This function fetches replenishment information for the current item at all stores that are supplied by the current warehouse. It then finds the forecasted ROQ for each store and distributes that quantity over any applicable simple packs. In this manner, it behaves similar to the main processing loop, but rather than calculating quantity to order, it is calculating the future 'need' for the item at the store. As each store is processed, a running total of the need for each pack is kept (by the `update_wh_need()` function). When all the stores have been processed, then the total need for each simple pack is used in calculating the warehouse's actual ROQ of that pack (in `get_repl_order_qty()`). Since the ROQ here is being calculated by simple pack rather than by item, there is no need to redistribute, and quantities returned are rounded up to a receivable pack size and the orders posted.

`update_wh_need()`

Updates the total need of a simple pack at a warehouse over all stores supplied by that warehouse. This is a support function for `replenish_to_wh()`.

`update_xdock_totals()`

Update the total quantity to order and the shortest lead time for all stores on a crossdock order. This information is used in placing the summary record of a crossdock order.

`place_xdock_wh_order()`

If a crossdock order is being created, a record must be written to ORD_TEMP to order the items to the warehouse for allocation to stores. This function adds this 'summary' record to the ORD_TEMP buffer (but not to the REPL_RESULTS buffer). If the structure fills up, both buffers will be posted to the database and reset to be refilled.

`place_item_orders()`

Adds a record to the appropriate structure(s) for insert to ORD_TEMP and/or REPL_RESULTS. The conditions for writing records on the ORD_TEMP is when the following were met. The order control must be 'Automatic' or 'Semi-Automatic'. And if SUP_INV_MGMT.due_order_process_ind is 'Y' or SUP_INV_MGMT.due_order_process_ind is 'N' and ROQ_PKG.due_ind is 'Y'. Records will be written to the REPL_RESULTS buffer, if the system-level Replenishment Results option is set to "Yes", or SUP_INV_MGMT.due_order_process_ind is 'Y' or ROQ_PKG.due_ind is 'Y'. If one or both structures fill up, they will be posted to the database and reset to be refilled.

`post_orders()`

Actually writes order information to the database, inserting to ORD_TEMP and REPL_RESULTS.

`update_review_date()`

Updates the replenishment information table (REPL_ITEM_LOC) to reflect the fact that item-location combinations have just been evaluated.

`get_repl_order_qty()`

Gets the recommended order quantity (ROQ) and other information for an item-location combination. Essentially just a wrapper for the REPLENISHMENT_SQL.GET_ITEM_LOC_REVIEW_TIME and GET_REPL_ORDER_QTY_SQL.REPL_METHOD stored PL/SQL procedures.

`get_domain_info()`

The GET_REPL_ORDER_QTY_SQL stored procedure requires a forecasting domain when performing calculations for item-location combinations using a forecasted method of replenishment (Dynamic or Time Supply). For a given department, class or subclass (depending on the system-level Domain Level option), this function finds the associated forecasting domain. The first time this is called, it will call `build_domain_cache()` to pull all domain information for this thread into an array, and return the desired domain. Subsequent calls will only query the array.

`build_domain_cache()`

Populates a structure with information on department/class/subclas/domain relationships. Because the program is threaded by department, the function will only pull domains associated with departments, classes, or subclass associated with the current thread.

`copy_repl_info()`

Copies replenishment information from the fetch array into a more convenient structure for evaluation and processing.

`copy_pack_to_repl()`

Copies replenishment and simple pack information into a `repl_info_struct` and a pack distribution structure for calculating ROQ and building orders for a warehouse. This is a support function for `replenish_to_wh()`.

`Get_vendor_line_info()`

Get vendor line constraint information from SUPS_INV_MGMT table. Copy them to `repl_info_struct`. They were needed by package `cal` for getting ROQ.

`size_repl_info_struct()`

Allocates memory for the elements of a replenishment information structure.

`size_ord_temp_struct()`

Allocates memory for the elements of an ORD_TEMP insert structure.

`size_repl_results_struct()`

Allocates memory for the elements of a REPL_RESULTS insert structure.

`size_pack_dist_struct()`

Allocates memory for the elements of a simple pack distribution structure.

size_xdock_info_struct()

Allocates memory for the elements of a crossdock totals structure.

final()

The normal Retek final function, this closes down the restart/recovery API.

Database Interaction

Tables Selected From:

- DOMAIN_CLASS
- DOMAIN_DEPT
- DOMAIN_SUBCLASS
- ITEM_SUPP_COUNTRY
- PACKHEAD
- PACKITEM
- PACKSTORE_HIST
- PERIOD
- RAG_SKUS_ST_HIST
- REPL_DAY
- REPL_ITEM_LOC
- STORE
- SYSTEM_OPTIONS
- WH
- WIN_STORE_HIST
- V_RESTART_DEPT (*View*)

Tables Inserted To:

- ORD_TEMP
- REPL_RESULTS

Tables Updated:

- REPL_ITEM_LOC

I/O Specification

N/A

Technical Issues

N/A

Automatic Replenishment Order Maintenance [rplprg]

Design Overview

This program looks at all approved replenishment orders created during the nightly batch run. These orders are compared with any vendor minimums that may exist. Orders that do not meet the vendor minimums are either deleted or placed in worksheet status. A flag held at the supplier inventory management level (SUP_INV_MGMT.ORD_PURGE_IND), determines what action is taken on orders that fail minimums.

Vendor minimums can be held at the order, sku, or location level. Order and location level minimums are held on the SUP_INV_MGMT table. There is a flag that determines if they are applied at the order level or at the location level. Vendor minimums at the sku level are held on the ITEM_SUPP_COUNTRY table.

When the SUP_INV_MGMT.ORD_PURGE_IND is 'N', a failure at any level causes the order to be placed in worksheet status. When the SUP_INV_MGMT.ORD_PURGE_IND is 'Y', a failure at the location level causes the offending location to be deleted, a failure at the sku level caused the offending sku to be deleted, and a failure at the order level caused the entire order to be deleted.

For any orders that fail vendor minimums when the SUP_INV_MGMT.ORD_PURGE_IND is 'Y', a record is written to the

SUPS_MIN_FAIL table for reporting purposes. This table is purged at the beginning of the program to remove old records.

After order records are purged, any applicable deals, brackets and allowances are applied to the orders. Open to buy is then updated for any orders built in approved status. If any orders are contract orders, the contract amounts are updated as well to reflect any order record deletions.

This program should run directly after the replenishment supcnstr program. It is important that this program runs before any other process affects the generated orders.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
ORDHEAD		Yes	No	Yes	Yes
ORDLOC		Yes	No	No	Yes
ORDSKU		Yes	No	No	Yes
DESC_LOOK		Yes	No	No	No
ORD_INV_MGMT		Yes	No	No	Yes
ORDSKU_DISCOUNT		No	No	No	Yes
DEAL_CALC_QUEUE		No	Yes	No	Yes
ITEM_SUPP_COUNTRY		Yes	No	No	No
REPL_RESULTS		No	No	No	Yes
SUPS_MIN_FAIL		No	Yes	No	Yes
ALLOC_HEADER		Yes	No	No	Yes
ALLOC_DETAIL		No	No	No	Yes
CONTRACT_HEADER		Yes	No	Yes	No
CONTRACT_DETAIL		Yes	No	Yes	No
OTB		No	No	Yes	No
ORDSKU HTS		Yes	No	No	Yes
REV_ORDERS		No	Yes	No	NO
PERIOD		Yes	No	No	No
SYSTEM_OPTIONS		Yes	No	No	No

Re-run:

If this program terminates abnormally, restart without recovery.

Scheduling Constraints

Processing Cycle: PHASE 3

Scheduling Diagram: Run after supcnstr.pc

Pre-Processing: N/A

Post-Processing: prepost.pc (prepost user/password rplprg post)

Threading Scheme:

Restart Recovery

```
EXEC SQL DECLARE c_order CURSOR FOR

    SELECT  ordhead.order_no,

            ordhead.contract_no,

            ordhead.payment_method,

            ordhead.supplier,

            NVL(ord_inv_mgmt.ord_purge_ind, 'N'),

            ordhead.import_order_ind,

            ordhead.dept,

            ord_inv_mgmt.ord_purge_ind

    FROM    ordhead,

            ord_inv_mgmt

    WHERE   ordhead.order_no = ord_inv_mgmt.order_no

           AND ordhead.order_type = 'ARB'

           AND ordhead.status = 'A'

           AND ordhead.orig_approval_id = 'Replen'

           AND ordhead.written_date = TO_DATE(:os_vdate, 'YYYYMMDD')

           AND ordhead.order_no > NVL(:os_order_no, -9999);
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Replenishment [rplrsprg.pc]

Design Overview

The replenishment extraction programs (rplext, reqext) write a number of records to the REPL_RESULTS table. This table holds records meant to track certain components of the replenishment algorithms that run during the batch process. Information like order points, safety stock is written to this table for every item/location processed against which the user can perform analysis. Over time, records on this table become unneeded and should be cleared out. The Replenishment Results Purge Program goes through the table and clears out those records that are older than a predetermined number of days.

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: No constraints; Phase V or ad-hoc.

Scheduling Diagram: N/A.

Pre-Processing: N/A.

Post-Processing: N/A.

Threading Scheme: N/A.

Restart Recovery

Driving Cursor

Because this program performs only deletes, there is no need for restart/recovery or multithreading, and there is no driving cursor. However, this program still needs an entry on RESTART_CONTROL to determine the number of records to be deleted between commits.

Program Flow

N/A.

Shared Modules

N/A.

Function Level Description

main

Since this program does not use the standard restart/recovery API, this main function will be different than most. It will simply log on to the database (using Retek's LOGON function), call init(), and call process(). Any errors returned from these functions should cause the program to write an error message to the daily log and rollback all work before exiting.

init

This function fetches the threshold date, which is the current processing date (PERIOD.vdate) minus the number of days to retain REPL_RESULTS records (SYSTEM_OPTIONS.repl_results_purge_days). This function should also fetch the commit_max_ctr from RESTART_CONTROL to determine how many records can be deleted between commits.

process

This program will, in a loop, delete from REPL_RESULTS where the repl_date is less than the threshold date and the rownum is less than the commit counter. If this statement processes no rows, then the table has been fully purged and the function should return successfully. Otherwise, it should commit and return to the top of the loop. The loop with intermittent commits is necessary in order to ensure that rollback segments are not in danger of overflowing.

I/O Specification

N/A.

Return to Vendor SQL Loader [rtv_upload.ctl]

Design Overview

The following program is a SQL Load script to load the Nautilus return to vendor upload file, rtv_upload.dat, into a staging table: lif_rtv

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: SQL loading must occur after Nautilus creates the rtv flat file, and it must occur before lifrtvup.pc.

Pre-Processing: N/A

Post-Processing: Sqlldr '.bad' files will be written with any data that cannot be inserted into the staging tables. Also, '.log' files will be created to view the results of the Sql load.

Threading Scheme: N/A

Program Flow

N/A

Function Level Description

N/A

I/O Specification

SQL Load the Logistics RTV upload file using the following control file (ie. rtv_upload.ctl):

Fields are specified by position in file, not by using delimiters.

LOAD DATA

INFILE 'rtv_upload.dat'

INTO TABLE lif_rtv

Location	POSITION(1:4)	INTEGER EXTERNAL
Batch_no	POSITION(6:12)	CHAR
RTV_order_no	POSITION(14:20)	INTEGER EXTERNAL
Item_id	POSITION(22:37)	CHAR
RTV_auth_id	POSITION(39:50)	CHAR
Unit_qty	POSITION(52:59)	INTEGER EXTERNAL
Container_qty	POSITION(61:66)	INTEGER EXTERNAL
Supplier	POSITION(68:74)	CHAR

To invoke SQL Loader user the following command line: sqlldr
userid/password@database control=rtv_upload.ctl log=rtv_upload.log

Technical Issues

N/A

Purge RTV [rtvpgr]

Design Overview

The RTV Purge (rtvpgr) module deletes out-dated RTV records from the system. RTV transactions that have been completed shipped or cancelled and those for which the completion date exceeds the RTV retention period are deleted. The RTV retention period is maintained on-line as a system option.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
RTV_HEAD	Yes	Yes	No	No	Yes
RTV_DETAIL	Yes	No	No	No	Yes
UNIT_OPTIONS	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE AD-HOC (monthly)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

```

SELECT rtv_head.rtv_order_no
FROM rtv_head
WHERE 0 < (SELECT NVL(MONTHS_BETWEEN(period.vdate,
                                rtv_head.completed_date), 0)
          - unit_options.rtv_order_history_months
FROM period, unit_options);

```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Return to Vendor SQL Loader [rtv_upload.ctl]

Design Overview

The following program is a SQL Load script to load the Nautilus return to vendor upload file, rtv_upload.dat, into a staging table: lif_rtv

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: SQL loading must occur after Nautilus creates the rtv flat file, and it must occur before lifrtvup.pc.

Pre-Processing: N/A

Post-Processing: Sqlldr '.bad' files will be written with any data that cannot be inserted into the staging tables. Also, '.log' files will be created to view the results of the Sql load.

Threading Scheme: N/A

Program Flow

N/A

Function Level Description

N/A

I/O Specification

SQL Load the Logistics' RTV upload file using the following control file (ie. rtv_upload.ctl):

Fields are specified by position in file, not by using delimiters.

LOAD DATA

INFILE 'rtv_upload.dat'

INTO TABLE lif_rtv

Location	POSITION(1:4)	INTEGER EXTERNAL
Batch_no	POSITION(6:12)	CHAR
RTV_order_no	POSITION(14:20)	INTEGER EXTERNAL
Item_id	POSITION(22:37)	CHAR
RTV_auth_id	POSITION(39:50)	CHAR
Unit_qty	POSITION(52:59)	INTEGER EXTERNAL
Container_qty	POSITION(61:66)	INTEGER EXTERNAL
Supplier	POSITION(68:74)	CHAR

To invoke SQL Loader user the following command line: sqlldr
userid/password@database control=rtv_upload.ctl log=rtv_upload.log

Technical Issues

N/A

[saaldnld.pc]

Design Overview

The saaldnld.pc needs to be modified to change the format of Unit Quantity, when downloading information to RDM. The new Unit Quantity field for the interface is now Number(12,4) opposed to the original Number(6).

Scheduling Constraints

No change to RMS 8.0

Restart Recovery

No change.

Program Flow

No change.

Shared Modules

No change.

Function Level Description

The saaldnld.pc needs to be modified to change the format of Unit Quantity, when downloading information to RDM. The new Unit Quantity field for the interface is now Number(12,4) opposed to the original Number(6).

I/O Specification

Stock Order file

File Header:

Field Name	Default Value	Field Type	Description
Record Type	FHEAD	Char(5)	Identifies file record type
Line Number	Incremented internally	Number(10)	Sequential line number
File Type	STOR	Char(5)	Identifies file
Create Date	YYYYMMDD HH24MISS format	Char(14)	File create date

File Detail:

Field Name	Default Value	Field Type	Description
Record type	FDETL	Char(5)	identifies the Detail line
Line number	Incremented internally	Number(10)	Sequential line number
Action_type	'A', 'M', , 'D'	Char(1)	Add, modify, or delete action type
Location	Alloc_header.w h	Number(4)	From Warehouse location
Transaction Date/Time	Sysdate	Datetime(12)	System date
Distribution Number	'A' + alloc_no	Char(9)	Allocation number. Prefix 'A' for alloc
Download Comment	NULL	Char(30)	Comment to be printed on the label (future use)
Pick-Not-Before Date	Alloc_header. release_date or vdate if NULL release_date	Char(8)	Date before which merchandise will not be distributed

Field Name	Default Value	Field Type	Description
Pick-Not-After Date	PNB date + (default period from codes table where code type = 'DEFT' and code = 'DATE')	Char(8)	Date by which merchandise must be distributed
Event code	NULL	Char(6)	Identifier of event
Event description	NULL	Char(25)	Event description
Priority	1	Char(4)	Priority
Order Type	UPPER(system_options.default_order_type)	Char(9)	Type of Order: AUTOMATIC, MANUAL or WAVE
Break By Distro	'N'	Char(1)	Controls the mixing of orders (distros) in a container
Carrier Code	NULL	Char(4)	Code of the carrier for the order
Carrier Service Code	NULL	Char(6)	Carrier's service code for the delivery, First Class, and son on (future Use)
Route	NULL	Char(10)	Route specified for the delievery
Ship Address Description	NULL	Char(30)	The description (such as the store name). This is the first line of the address block
Ship Address Line 1	NULL	Char(30)	Shipping Address Line 1
Ship Address Line 2	NULL	Char(30)	Shipping Address Line 2
City	NULL	Char(25)	Shipping City

Field Name	Default Value	Field Type	Description
State	NULL	Char(3)	Shipping State
Zip	NULL	Char(10)	Shipping Zip
Billing Address Description	NULL	Char(30)	The Description (such as company name). This is the first line of the address block.
Billing Address Line 1	NULL	Char(30)	Billing Address Line 1
Billing Address Line 2	NULL	Char(30)	Billing Address Line 2
Billing Address Line 3	NULL	Char(30)	Billing Address Line 3
Amount 1	0.00	Number(8,2)	Amount Charge 1
Amount 2	0.00	Number(8,2)	Amount Charge 2
Amount 3	0.00	Number(8,2)	Amount Charge 3
Po number	NULL	Char(9)	Unique identifier of the purchase order, prefixed with 'P'

File Trailer:

Field Name	Default Value	Field Type	Description
Record Type	FTAIL	Char(5)	Identifies file record type
Line Number	Incremented internally	Number(10)	Sequential line number
File Line Count	Total number of detail lines	Number(6)	Total number of transaction lines in file (not including FHEAD and FTAIL)

Stock Allocation file

File Header:

Field Name	Default Value	Field Type	Description
Record Type	FHEAD	Char(5)	Identifies file record type
Line Number	Incremented internally	Number(10)	Sequential line number
File Type	STAL	Char(5)	Identifies file
Create Date	YYYYMMDD HH24MISS format	Char(14)	File create date

File Detail:

Field Name	Default Value	Field Type	Description
Record Type	FDETL	Char(5)	identifies the Detail line
Line Number	Incremented internally	Number(10)	Sequential line number
Action_type	'A', 'M', , 'D'	Char(1)	Add, modify, or delete action type
Location	Alloc_header.w h	Number(4)	From Warehouse location
Transaction Date/Time	Sysdate	Datetime(12)	System date
Distribution Number	'A' + alloc_no	Char(9)	Allocation number. Prefix 'A' for alloc
Item ID	Alloc_header.sk u	Char(16)	Unique item identifier
Requested Unit Quantity	Alloc_detail.qty _allocated	Number (12,4)	Quantity allocated
Destination ID	Alloc_detail.sto re or wh	Number(4)	Allocation location
Price	Item_zone_pric e.unit_retail	Number(5,2)	Retail price
Print UPC Flag	'N'	Char(1)	Print upc flag (future use)

Field Name	Default Value	Field Type	Description
Ticket Type	Item_ticket.ticket_type_id or '0000'	Char(4)	Receiving Ticket type of item.
Priority	'1'	Char(4)	Priority (highest 1)
Expedite Flag	'N'	Char(1)	Flag indicating whether the order should be shipped via normal or expedite carrier service

File Trailer:

Field Name	Default Value	Field Type	Description
Record Type	FTAIL	Char(5)	Identifies file record type
Line Number	Incremented internally	Number(10)	Sequential line number
File Line Count	Total number of detail lines	Number(6)	Total number of transaction lines in file (not including FHEAD and FTAIL)

Component Ticketing file

File Header:

Field Name	Default Value	Field Type	Description
Record Type	FHEAD	Char(5)	Identifies file record type
Line Number	Incremented internally	Number(10)	Sequential line number
File Type	CPTT	Char(5)	Identifies file
Create Date	YYYYMMDD HH24MISS format	Char(14)	File create date

File Detail:

Field Name	Default Value	Field Type	Description
Record type	FDETL	Char(5)	identifies the Detail line
Line number	Incremented internally	Number(10)	Sequential line number
Action_type	'A'	Char(1)	Add, modify, or delete action type
Location	Alloc_header.wh	Number(4)	From Warehouse location
Transaction Date/Time	Sysdate	Datetime(12)	System date
Distribution Number	'A' + alloc_no	Char(9)	Allocation number. Prefix 'A' for alloc
Item ID	Alloc_header.sku	Char(16)	Unique item identifier
Destination ID	Alloc_detail.store or wh	Number(4)	Allocation location
Component Item ID	v_packsku_qty.sku	Char (16)	Item identifier of the component
Price	Item_zone_price.unit_retail	Number(5,2)	Retail price

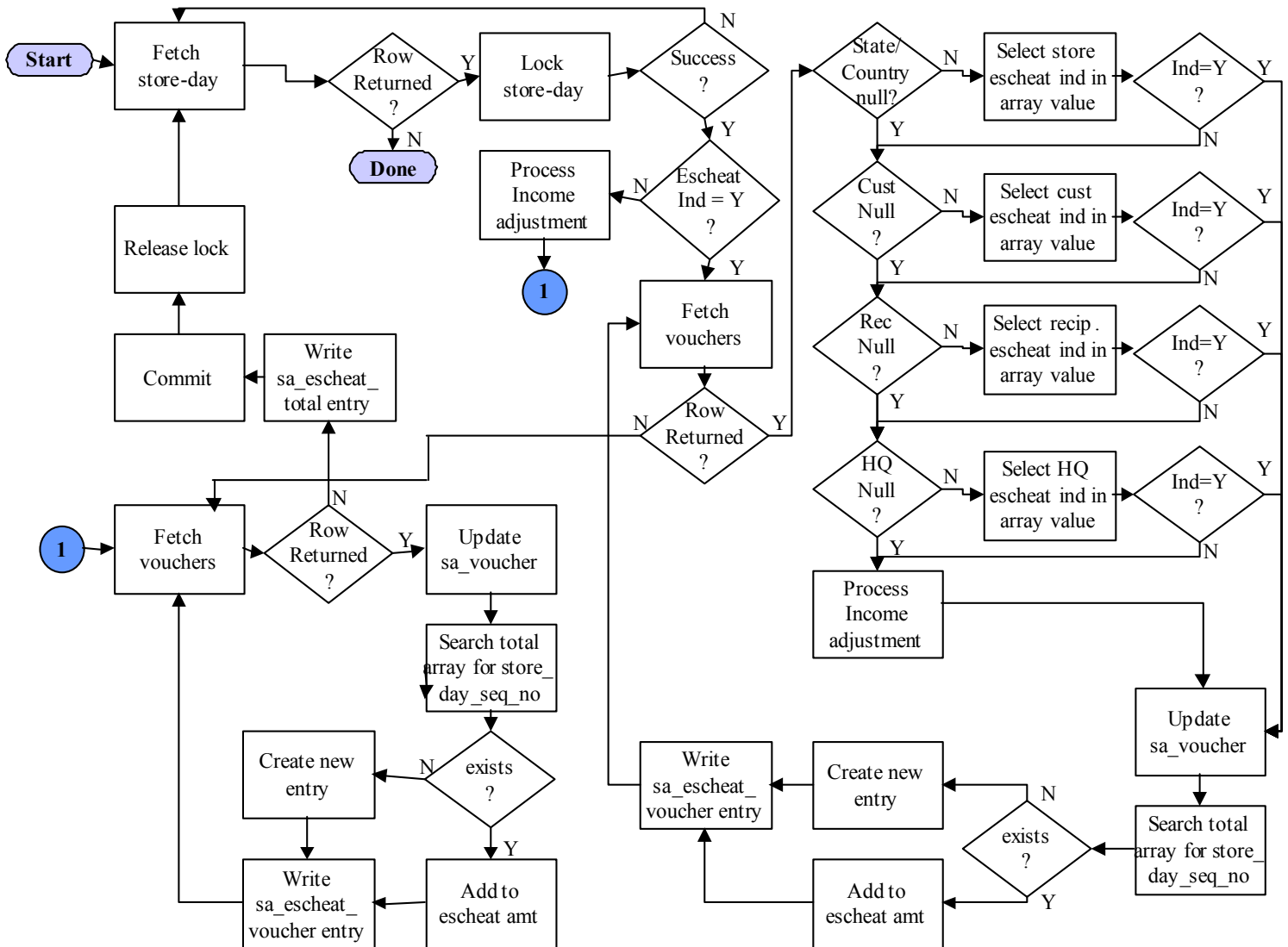
File Trailer:

Field Name	Default Value	Field Type	Description
Record Type	FTAIL	Char(5)	Identifies file record type
Line Number	Incremented internally	Number(10)	Sequential line number
File Line Count	Total number of detail lines	Number(6)	Total number of transaction lines in file (not including FHEAD and FTAIL)

[saescheat.pc]

Design Overview

New batch program that will set the status of vouchers that have met certain state's escheat rules, or have expired, to the proper status and produce a total for later export to Invoice Matching.



Stored Procedures / Shared Modules (Maintainability)

Libretek.a library functions:

Retek_init/retek_close – to initialize and close Retek’s restart/recovery module

FetchSysdate – to get the current date and time

FetchSaSystemOptions – to get ReSA system options

Libresa.a library functions:

Get_lock – to establish a read lock on a store-day

Release_lock – to release a store-day lock

Input Specifications

Table-To-Table

Foundation tables used by this program for reference include:

- Sa_escheat_options
- Sa_voucher_options
- Sa_system_options

Transaction data table used to drive the processing includes:

- Sa_voucher

Output Specifications

‘Table-To-Table’

Updates made to:

- Sa_voucher

Records inserted into:

- Sa_escheat_voucher
- Sa_escheat_total

Function Level Description

The logical unit of work in this program is a store, which will be used to drive the restart/recovery. Voucher entries will be selected based on the store that issued them. Totals will be produced using this store and the current vdate when the document was escheated.

Driving cursor:

```
SELECT      store_day_seq_no,
            store
FROM        sa_store_day,
            period
WHERE       business_date = vdate
ORDER BY    store
```

Init()

Call retek_init() to initialize restart recovery.

Get system date/time from fetchSysdate()

Get the escheat_ind from fetchSaSystemOptions()

Call OraNumInit() to initialize string numbers arithmetic operations.

Determine if escheatment is performed at the company level by calling IsCompanyStateEscheat().

Call FetchEscheatOptions() and populate an "Escheat Options" array with all records returned.

Allocate memory for any dynamic structures by calling SizeSDInfoArray(), SizeVoucherInfoArray(), and SizeUpdateSaVoucherArray().

Process()

Select a store-day to be processed by calling the first driving cursor.

Loop through this cursor

Attempt to lock the store-day by calling get_lock().

If escheat_ind is Yes, call ProcessEscheat()

Call DoUpdateSaVoucher() to force update of the database.

Else call ProcessIncomeAdjust()

Call WriteEscheatTotal ()

After pl_commit_max_ctr records have been fetched,

Call DoUpdateSaVoucher()

Call DoInsertEscheatVoucher()

Call retek_force_commit() to commit the database changes. This also releases the lock.

ProcessEscheat ()

Select tender_type_id,

voucher sequence number,

issued customer state,

issued customer country,

recipient state,

recipient country,

store state,

store country

from the sa_voucher, store, comphead and sa_voucher_options tables where the issued store equals the store in the current driving cursor record and the status is issued (I) and the expiration date is null and the issued date is not null and the (vdate – issued date) is greater than the escheat_no_days for that tender_type_id

Call the above cursor.

Loop through this cursor

If the issued state/country are not null, call PerformEscheat() passing these values and a 'state indicator' as parameters.

If successful, break

If the issued customer state/country are not null, call PerformEscheat() passing these values and a 'customer indicator' as parameters.

If successful, break

If the recipient state/country are not null, call PerformEscheat() passing these values and a 'recipient indicator' as parameters.

If successful, break

If the company state/country are not null, call PerformEscheat() passing these values and a 'company indicator' as parameters.

If successful, break

Call AdjustVoucher ()

PerformEscheat ()

Check the indicator passed to this function to determine which level to process at

Case 'state indicator'

Get value of the store_escheat_ind in the "Escheat Options" array for this state/country.

Case 'customer indicator'

Get value of the iss_cust_escheat_ind in the "Escheat Options" array for this state/country.

Case 'recipient indicator'

Get value of the recipient_escheat_ind in the "Escheat Options" array for this state/country.

Case 'company indicator'

Get value of the hq_escheat_ind in the "Escheat Options" array for this state/country.

If value is Yes, call EscheatVoucher(), passing the store_day_seq_no, voucher_seq_no, partner_type, partner_id and amount.

Else return 1

EscheatVoucher (store_day_seq_no, voucher_seq_no, partner_type, partner_id, amount)

Call UpdateSaVoucher(), passing voucher_seq_no and status of E.

Call InsertEscheatTotal(), passing E as the escheat_type, and the store_day_seq_no, partner_type, partner_id and amount.

Call WriteEscheatVoucher(), passing the escheat_seq_no and voucher_seq_no.

InsertEscheatTotal (store_day_seq_no, partner_type, partner_id,
amount,escheat_type (status))

Search “Escheat Total” array for store_day_seq_no, partner

If found, add to escheat amount.

Else select the next available escheat_seq_no by calling nextEscheatSeqNo() and insert the value into this array with the escheat_type passed in as a parameter, by calling InsertTotalNode().

Return escheat_seq_no.

WriteEscheatVoucher (escheat_seq_no, voucher_seq_no)

Insert escheat_seq_no, voucher_seq_no into sa_escheat_voucher array.

DoInsertEscheatVoucher()

This function does the actual insert to the sa_escheat_voucher table on the database.

UpdateSaVoucher(voucher_seq_no, status)

Insert voucher_seq_no, status and vdate into the sa_update_voucher array.

DoUpdateSaVoucher()

This function does the actual update of the sa_voucher table on the database. For each voucher_seq_no on the sa_update_voucher array, the status and the escheat_date get updated to the values in the array.

WriteEscheatTotal ()

Insert values from “Escheat Total” array into sa_escheat_total

ProcessIncomeAdjust ()

Select tender_type_id,

voucher sequence number,

issued customer state,

issued customer country,

recipient state,

recipient country,

store state,

store country

from the sa_voucher, store, comphead and sa_voucher_options tables

where the issued store equals the store in the current driving cursor record

and the status is issued (I)

and ((the expiration date is null

and the issued date is not null

and the (vdate – issued date) is greater than the escheat_no_days for that tender_type_id)

or the expiration date is less than the vdate)

Call the above cursor.

Loop through this cursor

Call AdjustVoucher () for each voucher returned from the cursor

AdjustVoucher ()

Call UpdateSaVoucher(), passing the voucher_seq_no and astatus of I

Call InsertEscheatTotal(), passing store_day_seq_no, partner_type, partner_id, amount, N as the escheat_type

Call WriteEscheatVoucher(), passing the escheat_seq_no and voucher_seq_no.

FetchEscheatOptions ()

Select all columns from the sa_escheat_options table into the “Escheat Options” array.

escheat_options_lookup (state, country)

This function returns a pointer to the element of the escheat_options array that contains the state and country passed in.

nodecmp ()

This function compares two elements in the escheat_options array.

InsertTotalNode ()

This function inserts a new element in the escheat_total array, using insertion sort.

escheat_total_lookup (store_day_seq_no, partner_type, partner_id)

This function returns a pointer to the element of the `escheat_total` array that contains the information passed in.

`totalcmp ()`

This function compares two elements in the `escheat_total` array.

`Size*Array` and `Free*Array` functions

These functions allocate memory for global arrays, such as the `escheat_options` and `escheat_total` arrays and free the memory allocated to these arrays.

`IsCompanyStateEscheat()`

This function populates a `company_level_data` array that holds information about whether the state where the HQ of the company is incorporated, has escheatment laws.

`Final ()`

Clean up any memory used.

Call `retek_close` to close restart recovery.

Free memory for any dynamic structures by calling `FreeSDInfoArray()`, `FreeVoucherInfoArray()`, `FreeUpdateSaVoucherArray()`, `FreeInsertEscheatVoucherArray()`, `FreeEscheatTotalArray()`, and `FreeInsertEscheatVoucherArray()`.

Scheduling Considerations

This program should be run as part of a monthly schedule. It should follow ReSA Totaling and Auditing process (`satotals.pc` and `sarules.pc`), and precede the export to Invoice Matching and the db purge (`sapurge.pc`).

Locking Strategy

Locking will be performed via the `get_lock` and `release_lock` library functions to lock the store-day during processing.

Restart/Recovery

A commit should be performed after processing is complete for each store-day.

[saexpach.pc]

Design Overview

This module will post Store/day deposit totals to the SA_STORE_ACH table and bank deposit totals for a given day to a standard BAI format file (version 2 only). The ACH export deviates from the typical Sales Audit export in that store/days must be exported by estimate even though errors may have occurred for a given day or store (depending on the unit of work defined).

Also, we are assuming that there is only one total to be exported for ACH per store/day.

No data is exported for an account if at least one store/day that belongs to an account does not have a store_status of 'C'.

Furthermore, the program estimates a 0 for a store/day that is closed, for example due to a holiday. An example is shown below (Wednesday is a holiday):

	Tues	Wed	Thu	Fri
est_deposit	0		10	
est_deposit_adj	5		15	0
Actual	10		15	10
Exported at close	5		25	0

In this example, we export only 5 (the adjustment) at close of Tuesday. The program is not run at close on Wednesday because it does not have a store_day_seq_no. Thus, on Thursday, the estimate for that day is 0 and the adjustment equals the actual. Also, on Thursday, we estimate that the total is going to be 10 and we export 25 at close of Thursday. Thus, the bank account should return to the minimum balance at this point.

Table	Operations Performed			
	Select	Insert	Update	Delete
Period	Yes	No	No	No
Sa_store_day	Yes	No	No	No
Sa_export_log	Yes	No	Yes	No
Sa_exported	No	Yes	No	No
Sa_store_ach	Yes	Yes	Yes	No
Sa_bank_ach	Yes	Yes	Yes	No
Sa_total	Yes	No	No	No
Sa_bank_store	Yes	No	No	No
Sa_store_day_write_lock	Yes	No	Yes	No
Sa_store_day_read_lock	Yes	No	No	No
Store	Yes	No	No	No
Partner	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Anytime – Sales Audit 9.0 is a 24/7 system.

Scheduling Diagram: This module should be run after the ReSA Totaling process: satotals and saaudit.

Threading Scheme: N/A

Restart Recovery

This module is in two distinct parts, with two different logical units of work. Thus restart/recovery has to be implemented so that the first part does not get reprocessed in case the program is being restarted. Details on the implementation follow.

The first driving cursor in this module retrieves a store/day to generate ACH totals. Once the first cursor is complete, the second retrieves bank locations by account numbers.

The first Logical Unit of Work (LUW) is defined as a unique store/day combination. Records will be fetched, using the first driving cursor, in batches of `commit_max_ctr`, but processed one store/day at a time.

The `libresa` library function `fetchStoreDayToBeExported` will drive the stores to be processed. Store/days are fetched sorted by store and then `business_date`. Before processing starts, a write lock is obtained using `get_lock()`. This driving cursor only fetches store/days with a `sa_export_log.status` of `SAES_R`. After a store/day is processed, `sa_export_log.status` is set to `SAES_P` so that this store/day will not be selected again if the program is restarted. We commit using `rettek_force_commit` after each store/day has been processed and `sa_export_log` updated, so as to release the lock.

In case a store/day could not be processed due to locking, then the store/day information is placed on a list (called locked store/day list) and the next store/day is processed. If the store for a store/day obtained from the first driving cursor, is on the locked store/day list, then this store/day cannot be processed. This is the case because there is a data dependency such that data from a particular store/day is dependent on data for the same store but at an earlier date. Thus, if a store/day cannot be processed, then subsequent store/days for the same store cannot be processed either. When `fetchStoreDayToBeExported()` returns a 1, that is there are no more store/days to process, then store/days on the locked store/day list are processed in turn. The program attempts to process each store/day on the list two more times. If the store/day is still locked, then it is skipped entirely and a message is printed to the error log.

The second LUW is a bank account number. Again, records will be fetched (this time using the second driving cursor) in batches of `commit_max_ctr`. However, a record does not necessarily represent an account number, i.e. more than one record may make up a single account number. This is the case because of the need to convert between store currency and bank currency (see function level description).

```
SELECT bs.partner_type, bs.partner_id, bs.bank_acct_no, bs.ach_acct_no,  
  
TOCHAR(sd.business_Date, 'YYYYMMDD'),  
  
sd.store_day_seq_no, s.currency_code, p.currency_code,
```

```

((NVL(sa.est_deposit,0) + NVL(sa.est_deposit_adj,0) +
NVL(sa.manual_deposit_adj,0)) * 10000)

FROM sa_bank_store bs,

      sa_store_ach sa,

      sa_export_log el,

      sa_store_day sd,

      store s,

      partner p

WHERE bs.store = sd.store

AND   bs.partner_type = sa.partner_type

AND   bs.partner_id = bs.partner_id

AND   bs.ach_acct_no IS NOT NULL

AND   s.store = sd.store

AND   p.partner_type = sa.partner_type

AND   p.partner_id = sa.partner_id

AND   sd.store_day_seq_no = el.store_day_seq_no

AND   sa.stpre = sd.store

AND   sa.business_date = sd.business_date

AND   el.status = :SAES_P /* Processed in the first part */

ORDER BY   bs.partner_type, bs.partner_id, bs.ach_acct_no, sa.business_date

```

Program Flow

Please see the following document for the complete structure chart of the standard export for ReSA.

Functional Design – SA export.doc

Shared Modules

retex library functions:

retex_init() – This function initializes restart/recovery.

retex_close() – This function cleans up restart/recovery.

retex_force_commit() – This function commits any change to the database.

Sales/Audit library functions (libresa):

fetchVdate() – This function is used to get the vdate.

fetchSysdate() – This function is used to get system date and time

fetchStoreDayToBeExported() – This function contains the first driving cursor.

get_lock() – This function is used to lock the store/day being processed.

OraNumInit() – Initialize OraNum functions.

OraNumAdd() – Add two large numbers passed in as strings.

OraNumSub() – Subtract two large numbers passed in as strings.

OraNumDiv() – Divide two large numbers passed in as strings.

Function Level Description

Init ()

Get the vdate from the period table and the system time.

Initialize restart/recovery by calling restart_init().

Process ()

Get the next store/day to be processed (exported) by calling fetchStoreDayToBeExported

Attempt to lock the store/day with a call to get_lock(). If this fails, write the store to a linked list (which contains all unprocessed store/days).

Skip to step 7 if the store of the store/day to be processed is for a store which is on the linked list.

Call the function postStoreACH for the current store day.

Set sa_export_log.status to SAES_P for the current store/day, so that it will not be processed again in case of a restart.

Call retek_force_commit() to commit changes to the database and to release write lock.

Loop from beginning until the return result of the function fetchStoreDayToBeExported = 1. A return value of 1 means that there are no more store/days to be exported.

Try to process the store/days that were not processed, that is, those that are on the linked list. Thus, for each store/day on the linked list, we try to obtain a lock. If one is not obtained, then we skip this store/day. If a lock is obtained, then we remove the store/day from the list.

Skip to step 12 if the store of the store/day to be processed is for a store which is on the linked list.

Call the function postStoreACH for the current store day.

Set sa_export_log.status to SAES_P for the current store/day, so that it will not be processed again in case of a restart.

Loop through steps 8 to 10, until each store/day in the list has been looked at.

Loop through steps 8 to 12 NUM_LOCK_RETRIES times. NUM_LOCK_RETRIES is by default 2. Thus, we try to attempt to process store/days that are locked two more times before giving up and skipping all locked store/days entirely.

For each store/day that was not processed, we write an error to the log.

Call the function postBankSummaryTotals.

Final ()

Clean up restart/recovery by calling retek_close().

postStoreACH ()

This function will generate and post an estimate and adjustment to the SA_STORE_ACH table for a given store day. The function postStoreACH will accomplish the following processes in the following order:

Get the following pieces of data for the system code SYSE_ACH: 1. the total for the current business date, 2. get the total for the following business date if it exists (by calling GetTomorrowTotal), 3. call the function GetPastData to get the totals for the past 4 weeks and for yesterday (that is, if the current store/day is for a Tuesday, then we want to get the totals for the past 4 Wednesdays and for yesterday). The latter pieces of data are obtained from the sa_store_ach table, by summing the estimate for a day with the adjustment for the same day. 4. call the function GetPartnerInfo to get partner type and partner id information.

If there are more than one total for SYSE_ACH for a particular day, then this should be noted in the error log. We expect only one total per store/day.

Call the function CalculateData to compute the estimate for the next business day and adjustment for the current store_day.

Call the function PostStoreACHTable.

GetTomorrowTotal ()

This function attempts to get the total for the next business day to be used as the estimate. It returns a -1 if a fatal error occurred, a 0 if it was able to get the total and a 1 if it was not able to get the total. If a store/day is never opened (i.e. a holiday), then a 0 is estimated for that store/day.

GetPastData ()

This function retrieves totals for the past 4 weeks and for the previous business day.

GetPartnerInfo ()

This function retrieves partner type and partner id for a store.

CalculateData ()

This function calculate the estimate for the next business day and adjustment for the current store_day.

Find the estimate for the following business date using the following rules:

If the total for the following business date exists, then this is the estimate.

Otherwise, the estimate is the average for the data for the past 4 weeks. If we obtain data for fewer than 4 weeks, then we use the available data, but if we do not obtain any data, then we use the current day's total as the estimate.

If the estimate is a 0, then we use the current day's total as the estimate.

Calculate the adjustment, which is the difference between the current date's total and the estimate for the current date (which lies on the row for the previous day on the sa_store_ach table).

PostStoreACHTable ()

This function inserts data into the sa_store_ach table. It updates if there is already an entry for the store, business date and partner.

If there is no entry in the sa_store_ach table for the current store day.

Create an entry in the SA_STORE_ACH table with the current store_day_seq_no and the new estimate and adjustment deposits for the current store_day_seq_no.

If there is an entry in the sa_store_ach table for the current store day.

Update the entry in sa_store_ach with the estimated deposit, and estimated deposit adjustment.

postBankSummaryTotals()

This procedure will summarize the bank transaction totals to the BAI output file. Please see the section on I/O specifications for more information about the format of this file.

Open and fetch from the second driving cursor.

If any entries are to be made (there are results from the cursor), create BAI file and write file header.

For each Bank location (partner_id), create a new Group Header record in the BAI file.

For each new account in the current bank create a new Account header record in the BAI file.

Call the function PostBankACHTable.

Keep running totals for bank location ACH amount, bank accounts ACH amount, and total ACH amounts.

If the account changes for the same bank, create an account trailer record for the above bank and account, then repeat loop back to step 4 with the new account.

If the bank location changes (partner_id).

Create a group trailer record for the previous bank location.

Create a new Group header record for the new bank location.

Continue looping back to step 1 until there are no more bank-locations to process.

Write BAI file tail record and save file.

Close the Bank cursor.

Mark all store days that were not locked (i.e. those with a sa_export_log.status of SAES_P) as completed (SAES_E) in the sa_export_log.

postBankSummaryTotals()

This function inserts into the table sa_bank_ach. It updates if there already exist a record for the same partner and business date.

1. If an entry does not exist for the current bank and date in the sa_bank_ach table:

Make an entry in the sa_bank_ach table for the current bank and account placing the sums of the store ACH amounts and adjustments in the ACH amount field (sa_bank_ach.ach_amt).

Create a Summary detail record in the BAI file with the Appropriate Type Code (See the BAI standard output file format for more details). For the *amount* field, use the current bank ACH deposit (sa_bank_ach.ach_amt) for the current bank and account.

2. If an entry exists for the current bank and date in the sa_bank_ach table:

Update the sa_bank_ach table with the bank ACH deposit amount (sa_bank_ach.ach_amt).

Create a Summary detail record in the BAI file with the Appropriate Type Code (See the BAI standard output file format for more details). For the amount field, add the manual deposit adjustment (sa_bank_ach.manual_ach_adj) from the current bank ACH deposit (sa_bank_ach.ach_amt) for the current bank and account.

IsAccountOk ()

This function is used to determine whether an account can be exported to ACH. If a store has not been 'P'rocessed in the first part, then its account cannot be exported. For example, if stores 1035, 1036 and 1037 belong to the same account, but store 1037 has not been processed, then this account cannot be exported.

MarkAllStoreDaysCompleted ()

This function sets the sa_export_log.status to 'E' for stores whose accounts have been exported. That is, if a store belongs to an account that was not exported, it will not be marked 'E'. This cursor has a subquery which works in a similar way as the one in IsAccountOk().

SetCurrencyDecimals ()

Given a currency code and an amount with 4 implicit decimals, this function will give out an amount with the appropriate number of decimals for the currency. For more details, see the BAI file format documentation. For example, there are two implicit decimals for the US Dollar, but none for the Japanese Yen. This function may need to be expanded because only a select few currencies are being processed. The last two decimal places are dropped for currencies that are not explicitly defined.

I/O Specification

BAI Version 2 Interface File

The BAI version 2 file format is specified below, according to the ReSA ACH export requirements. All fields will be delimited by a comma and records by a '/'.

File Header

Field Name	Field Description	Default Value
Record Code	01	01
Sender Identification	Transmitter of file	i.e. Tosco
Receiver Identification	Recipient of file	This is the routing number and account number of the consolidating account.
File Creation date	YYMMDD format	
File Creation Time	HHMM format	
File Identification Number	Must be new for each creation date.	01
Physical Record Length	Optional	Leave blank - Variable

Field Name	Field Description	Default Value
Block Size	Optional	Leave blank - Variable
Version Number	2	2
End of record Delimiter	/	/

Sample file Header:

01,122099999,123456789,990630,0200,1,,,2/

Tosco (122099999) is sending data to Melon Bank (123456789). The file was created June 30, 1999 at 2:00pm. This is the first file created on this date (1). Both file length and block sizes are set to their defaults (noted by adjacent delimiters), and we are using version 2 BAI file format. The slash indicates the end of the file header record.

Group Header

The group header will identify a bank location where existing store accounts exist.

Field Name	Field Description	Default Value
Record Code	02	02
Ultimate Receiver Identification	Optional. Final Receiver of this group of data.	This is the routing number and account number of the consolidating account.
Originator Identification	Originator Id	Creator of the file (i.e. Tosco)
Group Status	1 = Update 2 = Deletion 3 = Correction 4 = Test only	1
As-of-date	YYMMDD	
As-of-time	Optional - HHMM	
Currency Code	Optional	USD – US is default

Field Name	Field Description	Default Value
As-of-date Modifier	Optional 1 = Interim previous-day data 2 = Final previous-day data 3 = Interim same-day data 4 = Final same-day data	Does not affect processing, reference only Leave blank.
End of record delimiter	/	/

Sample Group header:

02,031001234,122099999,1,990630,0230,,/

Norwest bank (031001234) is receiving data from Tosco (122099999). The data in the group has a status of update and the data are as-of-date June 30, 1999 at 2:30pm. The currency code is defaulted to US dollars. We have not specified an as-of-date modifier for this data.

Account Identifier and Summary Status

The account identifier and summary status will accomplish two functions. First, it will identify the account number at the bank location specified in the group header above. Second, it will specify the ACH debit amount as a summary status.

Field Name	Field Description	Default Value
Record Code	03	03
Customer Account Number	Customer account number at originator financial institution.	i.e. Tosco's bank account number at Norwest Minneapolis branch.
Currency Code	Optional	USD – US is default
Type Code	Optional – identifies the type of summary or status data.	450 – specifies Total ACH Debits
Amount	Optional – Signed amounts (+) or (-), default is positive. Expressed without a decimal	

Field Name	Field Description	Default Value
Item Count	Optional – No implied decimal	Default is unknown Leave blank
Funds type	Optional	Default is unknown Leave blank
End of record delimiter	/	/

Sample Account Identifier and Summary status record:

03,741727229,450,1500000,./

This line identifies Tosco's bank account (741727229) at the Norwest bank in Minneapolis. We have a summary Total ACH debit amount of \$15,000.00. All other fields are left to their defaults.

Account Trailer

Field Name	Field Description	Default Value
Record Code	49	49
Account Control Total	Algebraic sum of all "amount" fields in the preceding type 03 and 16 type records associated with that account. Signed amount with default to positive.	
Number of records	Integer. Total number of records in the account, including the 03, 16 and this 49-type record.	
End of record delimiter	/	/

Sample account trailer record:

49,1500000,2/

Here we have an account total of \$15,000.00 consisting of 2 records (including this 49 record).

Group Trailer

Field Name	Field Description	Default Value
Record Code	98	98
Group Control Total	Algebraic sum of account totals in this group. This field includes the sign (+) or (-). The default is positive if no sign is specified.	
Number of accounts	Integer. The number of 03 records in this group.	
Number of records	Integer. The total number of all records in this group. Include all 02, 03, 16, 49, 88 and this 98 record.	
End of record delimiter	/	/

Sample Group trailer record:

98,18000000,2,6/

This example shows a total for all accounts in this group as \$180,000.00. Two accounts and 6 records are included in this group.

File Trailer

Field Name	Field Description	Default Value
Record Code	99	99
File Control Total	Algebraic sum of all group control totals in this file. This field includes the sign (+ or -). If no sign precedes the total, the default is positive.	
Number of groups	Number of 02 records in this file.	
Number of records	Total number of records in this file. Include this 99 record.	
End of record delimiter	/	/

Sample File trailer record:

99,19500000,2,10/

This example shows a total group total amount of \$195,000.00. There were two groups and 10 records for the entire file. (Including this 99 record.)

[Saexpgl107.pc]

Design Overview

This module will post all properly configured user defined ReSA totals to the User defined General ledger application (Oracle or PeopleSoft). Totals without errors will be posted to the appropriate accounting ledger, as defined in the Sales Audit Oracle cross-reference user module. Depending on the unit of work system option, the data will be sent at either the store day or individual total level. Newly revised totals that have already been posted to the ledger will have their previous revision reversed, and the new total posted to the appropriate accounts. Transactions that are from previous periods will be posted to the current period.

This version of the program is meant for the interface between RMS 9.0 and Oracle Financials 10.7.

Table	Operations Performed			
	Select	Insert	Update	Delete
Period	Yes	No	No	No
Sa_system_options	Yes	No	No	No
Sa_store_day	Yes	No	No	No
Sa_export_log	Yes	No	Yes	No
Sa_error	Yes	No	No	No
Sa_error_cat	Yes	No	No	No
Sa_store_error	Yes	No	No	No
Sa_exported	Yes	Yes	No	No
Sa_total_value	Yes	No	No	No
Sa_total_usage	Yes	No	No	No
Sa_balance_group	Yes	No	No	No
Fif_ora_gl_setup	Yes	No	No	No
sa_Fif_ora_cross_ref	Yes	No	No	No
Sa_error_rev	Yes	No	No	No
Sa_exported_rev	Yes	No	No	No
Sa_total_value_rev	Yes	No	No	No
Sa_store_day_lock	Yes	Yes	No	Yes
GL_INTERFACE	No	Yes	No	No
PS_CPI_GL_DATA	No	Yes	No	No

Table	Operations Performed			
	Select	Insert	Update	Delete
Fif_ps_coa_setup	Yes	No	No	No

use a new table sa_fif_ora_cross_ref,

sa_Fif_ora_cross_ref		
Location	NOT NULL	NUMBER(4)
Total_id	NOT NULL	VARCHAR2(10)
rollup_level_1	NOT NULL	NUMBER(3)
rollup_level_2	NOT NULL	NUMBER(3)
rolup_level_3	NOT NULL	NUMBER(3)
DR_sequence1-10	NULL	VARCHAR2(25)
DR_CCID	NOT NULL	NUMBER(15)
CR_sequence1-10	NULL	VARCHAR2(25)
CR_CCID	NOT NULL	NUMBER(15)

PK_SA_FIF_ORA_CROSS_REF

Column

ROLLUP_LEVEL_1 Default value: -1

ROLLUP_LEVEL_2 Default value: -1

ROLLUP_LEVEL_3 Default value: -1

LOCATION Default value: -1

TOTAL_ID

If any of the primary key columns are not selected, a value of -1 will be inserted to ensure integrity on the table. The -1 value represents a wildcard for that field.

Scheduling Constraints

Processing Cycle: Anytime – Sales Audit 9.0 is a 24/7 system.

Scheduling Diagram: This module should be run after the ReSA totaling process: satotals.pc and sarules.pc.

Threading Scheme: N/A

Restart Recovery

The logical unit of work for this module is defined as a unique store/day combination. Records will be fetched, updated and inserted in batches of `pl_commit_max_ctr`. Only one commit will be done: at the end, after a store/day has been completely processed, a call to `release_lock()` performs a commit.

There are 2 driving cursors in this module. The first picks a store/day to work on. The second fetches the totals to be posted for the store/day.

Driving cursor 1:

This driving cursor is embedded in the library function `fetchStoreDayToBeExported()`. Given a system code, of 'SYSE', this function fetches all store/days with a `store_status` of 'C'lose, a `data_status` of 'F'ully loaded and an `audit_status` of 'A'udited, 'S'tore errors pending or 'H'Q errors pending that are ready to export to the given system.

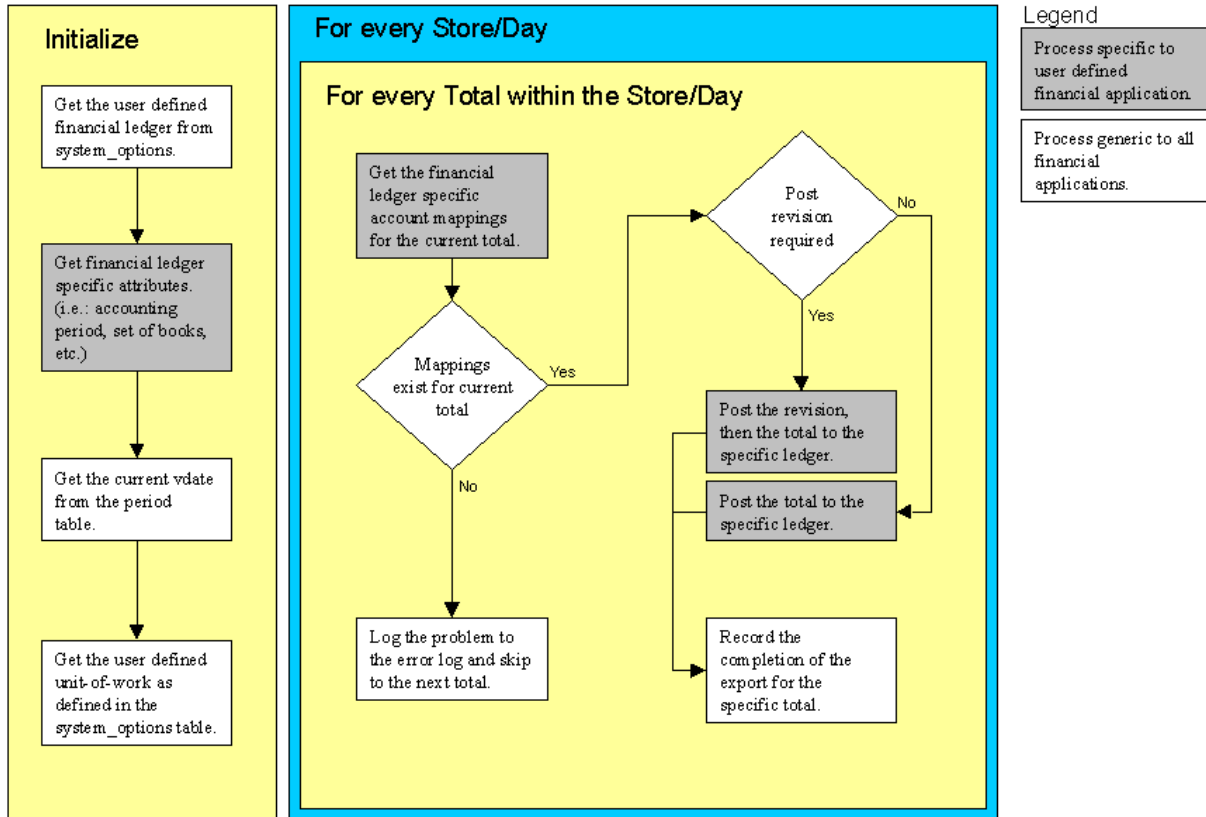
Driving cursor 2:

This driving cursor is embedded in the library function `getTotal()`. Given a `store_day_seq_no` and a usage type of 'SAYT', this function retrieves all totals.

Program Flow

Structure Chart

Below is a simple flow of the general ledger export and its generic and financial application specific modules:



Shared Modules

libresa.a – ReSA Library

get_lock – used to establish a read lock on a store/day.

release_lock – used to release a store/day lock.

fetchStoreDayToBeExported – This fetches all store days that are ready for export for a given usage type.

getTotal – This fetches all totals that can be exported for the given usage type and for the given store day.

fetchStoreDayErrorCount – This functions returns the number of errors pending for a given store day.

markTotalExported – records the passed total as exported

markStoreDayExported – records the passed store day as exported

fetchSaSystemOptions – This function retrieves all entries in the sa_system_options table.

getVdate – This function retrieves the vdate from the period table.

Refer to the following documents for more details on the export library:

Library Design – saexplib.doc.

libretek.a – Retek Library

rettek_init – initialize restart/recovery.

rettek_close – finalize restart/recovery.

Function Level Description

init ()

Call rettek_init.

Get the user financial application type from system_options.financial_app.

‘O’ = Oracle GL

‘P’ = PeopleSoft GL

Get the Financial application specific attributes (i.e. accounting period information, set of books identifier, etc.)

If Oracle GL, retrieve the following details as defined in the RMS database:

Fif_ora_gl_setup.set_of_books_id

Fif_ora_gl_setup.last_update_id

If Oracle GL, get the current period details from the Oracle GL table gl_period:

SELECT period_num, year_start_date, period_name, start_date, end_date

FROM gl_periods

WHERE vdate IS BETWEEN start_date AND end_date

AND adjustment_period_flag = ‘N’

Get the current vdate from the period table, using getVdate.

Get and save the value of sa_system_options.unit_of_work, by calling the function fetchSaSystemOptions.

process ()

Retrieve a store/day by calling fetchStoreDayToBeExported.

Attempt to lock the store/day with a call to `get_lock`. If this fails, go on to the next store/day.

Find out the number of errors pending for the store/day by calling `fetchStoreDayErrorCount`.

If the unit of work is store and the number of errors in the store/day is greater than zero, then release the lock by calling `release_lock` and skip the store/day, otherwise continue.

Retrieve a total to export by calling `getTotal`.

If Oracle GL, check to ensure that the selected total has a user defined cross-reference in the `sa_fif_ora_cross_ref` table by calling the function `getOracleMapping`. If a mapping (Oracle CCID) does not exist for the selected total log the problem in the Retek error log and go onto the next total.

If the `tran_sign` is 'N' (`code_type` is SAFD), the current retrieved value will be post to Oracle with negative sign.

Post the current total to the GL by calling the financial application specific function:

if Oracle, call `postOracleGL`.

If there are more totals for the selected store/day, loop through the store day totals (`getTotal`).

Call the library function `markStoreDayExported`.

Call `release_lock` and go on to the next store/day.

`final()`

Clean up – free any memory used.

Call `retek_close`.

`CanProcess()`

calling `fetchStoreDayErrorCount` to find out the number of errors pending for the store/day.

If the unit of work is store and the number of errors in the store/day is greater than zero, skip the store/day and write to the `if_errors` for the store/day.

`ProcessStoreDay()`

Get all the totals for the store/day by calling `getTotal()`.

For each `Total_id`, call `getOracleMapping()` for Oracle account.

If Status returned from getTotal() is 'N'. The opposite amounts will be posted to the GL_interface table (that is, send a negative number).

Call UpdateGLArray() to populate gl_data_array for inserting gl_interface table.

Call the library function markTotalExported and include the current period number. This function has to be called once for each total that is exported.

getOracleMapping()

This function will load local variables with the user-defined accounts and CCID's for the selected total/location combination from the SA_FIF_ORA_CROSS_REF table. If no results are returned, the total should be skipped with the appropriate message in the Retek error log.

UpdateGLArray()

This function write store/day total to the gl_data array for inserting to gl_interface.

Post the current total using the mapped local variables retrieved from the getOracleMapping function. First insert a record for the debit side of the transaction, then insert a record for the credit half of the transaction. (See GL_INTERFACE details below).

AddToList()

Setup linked list to hold locked store/day for later process.

GetNext()

This function move the pointer to the next unprocessed store/day.

RemoveFromList()

This function remove processed store/day from linked list.

DeleteList()

This function delete linked list, and free the momery.

postOracleGL()

This function will post the current total to the Oracle GL.

The following is a detailed explanation of the required columns in the Oracle GL_INTERFACE table.

GL_INTERFACE column explanation

status – This column represents the type of posting being applied. All inserts from this module, status should be set to 'NEW'.

set_of_books_id – This column represents the identifier for the book of accounts that this module will be posting to. This field should always be set to the value found in FIF_ORA_GL_SETUP.SET_BOOKS_ID

accounting_date – The date of the transaction/total –
SA_STORE_DAY.BUSINESS_DATE.

currency_code – The default system currency code

date_created – period.vdate

created_by – This field represents the identifier of the application/user whom created this journal entry. This value should be populated with the FIF_ORA_GL_SETUP.LAST_UPDATED_ID.

Actual_flag – The hard-coded value ‘A’ will represent actual amounts.

user_je_category_name – Journal entry source name for the posted transaction. This entry must exist in the Oracle USER_JE_CATEGORY_NAME column in the Journal Categories table prior to posting data to the GL. This value should be hard-coded to ‘ReSA’.

user_je_source_name - Journal entry source name for the posted transaction. This entry must exist in the Oracle USER_JE_SOURCE_NAME column in the Journal Sources table prior to posting data to the GL. This value should be hard-coded to ‘ReSA’.

currency_conversion_date – The date in which the total was converted to the default currency code. This value should be populated with the store day bussiness date.

User_currency_conversion_type – This value should be hard-coded to ‘Spot’.

Segment1 – 10 – These columns should be populated with either the debit segment values or the credit values (depending on which half of the total you are posting).

Entered_dr – If you are entering the debit half of the total, place the total amount in this column. If you are representing the credit half of the total, place a 0 in this column.

Entered_cr – If you are entering the credit half of the total, place the total amount in this column. If you are representing the debit half of the total, place a 0 in this column.

Reference21-22 – These columns will be hard-coded to NULL for the total exports.

Code_combination_id – If this is the debit half of the total adjustment, place the SA_FIF_ORA_CROSS_REF.DR_CCID. If this is the credit half of the total adjustment, place the SA_FIF_ORA_CROSS_REF.CR_CCID.
postPeopleSoftGL()

SizeGLData_array()

This function alllocate momery for gl_data_array.

I/O Specification

There are no input or output files for this export. All data is retrieved from ReSA database tables (as listed above) and posted to the Oracle GL staging table GL_INTERFACE or the PeopleSoft staging table PS_CPI_GL_DATA.

Sales Audit Export to Invoice Matching [saexpim.pc]

Design Overview

The purpose of this program is to support invoices from Direct Store Delivery and Escheatment sales audit transactions. Direct Store Delivery invoices refer to products or services that are delivered to the store and paid out at the store. This program will take DSD invoices that have been staged to the SA_TRAN_HEAD table by the saimptlog.pc program and move them into the INVC_HEAD table. All DSD transactions will be assumed paid. They can be assumed received if there is a proof of delivery number listed on them. Transactions with a vendor invoice ID or a proof of delivery number should be matched to any existing invoice in INVC_HEAD, and that invoice updated with the new information being interfaced. Invoices that do not match an existing invoice in INVC_HEAD will need to be inserted. Each transaction will be exported to INVC_HEAD table only once.

The Sales Audit Transaction type used to identify invoices for Direct Store Delivery transactions will be "Paid Out". Transaction types are stored on the codes tables with a code_type = 'TRAT'. The Paid Out transaction has a code of 'PAIDOU'. The Sales Audit sub-transaction types will be used to identify whether the invoice is an "Expense Vendor Payout" or a "Merchandise Vendor Payout". These types are stored on the codes table with a code_type = 'TRAS'. The codes will be 'EV' for Expense Vendor Payout and 'MV' for Merchandise Vendor Payout. Any Paid Out transaction with a sub transaction type of Expense Vendor will create a non-merchandise invoice and cause a record to be written to the INVC_NON_MERCH table. ReSA will store non-merchandise codes in the reason_code field on sa_tran_head. Valid values for these reason codes should correspond to the codes stored on the non_merch_code_head table.

In addition to DSD invoices, this program will also interface Escheatment totals to Invoice Matching. Escheatment is the process where an unredeemed gift certificate/voucher or credit voucher will, after a set period of time, be paid out as income to the issuing Retailer or in some states, the State receives this escheatment income. ReSA will be the governing system that determines who receives this income, but Invoice Matching will send the totals, with the related Partner, to Accounts Payable. Escheatment information will be stored on the ReSA SA_TOTALS table and will be used to create non-merchandise invoices in Invoice Matching. These invoices will be assumed not paid.

To accommodate Escheatment, a new calculation should be added to Sales Audit to create escheatment totals. ReSA automatically totals sales transactions based on calculation definitions that the customer's users have previously created using the online wizard. Whenever users create new calculation definitions or edit existing ones, they become part of the automated totaling process the next time that satotals.pc runs.

Table	Operations Performed			
	Select	Insert	Update	Delete
Period	Yes	No	No	No
Sa_system_options	Yes	No	No	No
Sa_export_options	Yes	No	No	No
Sa_store_day	Yes	No	No	No
Sa_store_day_read_lock	Yes	Yes	No	Yes
Sa_export_log	Yes	No	Yes	No
Sa_tran_head	Yes	No	No	No
Sa_tran_tender	Yes	No	No	No
Sa_exported	No	Yes	No	No
Sa_exported_rev	No	Yes	No	No
Sa_total	Yes	No	No	No
Sa_error	Yes	No	No	No
Invc_head	Yes	Yes	Yes	No
Invc_non_merch	Yes	Yes	Yes	No
Terms	Yes	No	No	No
Currency_rates	Yes	No	No	No
Addr	Yes	No	No	No

Stored Procedures / Shared Modules (Maintainability)

libretek library functions:

- retek_init/retek_close/retek_refresh_thread - to initialize and close Retek's restart/recovery module

libresa library functions:

- fetchSysdate – to get the current date and time
- fetchSaSystemOptions – to get ReSA system options
- fetchStoreDayErrorCount – to determine whether there are errors for the current store_day
- markStoreDayExported – to mark a store day as updated

- updateStoreDayExported – to force commit of status ‘E’ on sa_export_log for store days that have been exported
- getTotal – to fetch totals for escheatment invoices
- get_lock – to establish a read lock on a store_day
- release_lock – to release a store_day lock

Packages:

- INVC_SQL.NEXT_INVC_ID—generates a new invoice id
- DIRECT_STORE_INVC_SQL.CHECK_INVC_DUPS – Checks if invoice exists.

Input Specifications

Table-To-Table

Foundation tables used by this program for reference include:

- Sa_store_data
- Sa_system_options
- Sa_export_options
- Sa_error_codes
- Sa_error_impact

Transaction data tables used to drive the processing includes:

- Sa_tran_head
- Sa_total
- Sa_tran_tender
- Sa_exported
- Sa_error

Sa_total_usage – Add new Invoice Matching code ('IM') to the code_detail table where code_type = 'SAUT' (Sales Audit Usage Type). This code will be used as the invoice matching usage_type. Using the ReSA on-line wizard, add a new 'Total' calculation for escheatment transactions.

Total_id	Usage_type
ESCHEAT	IM

Sa_store_data – add invoice matching code to code_detail table where code_type = 'SYSE'

Code_detail

Code_type	Code	Code_desc	Code_seq
SYSE	IM	IM Export	7

Sa_store_data – For each store that will require the Invoice Matching Export, add a record to the sa_store_data table with a system_code of 'IM' (Invoice Matching) and imp_exp of 'E' (Export).

Store	System_code	Imp_exp
e.g. 1009	IM	E

Sa_export options – add IM code to the export options table to identify this new export to Invoice Matching. Since this program will not follow full disclosure logic, the multiple_export_ind should be 'N' (No).

System_code	Multiple_export_ind	Exp_detail_ind
SYSE	N	IM Export

Output Specifications

Table-To-Table

Updates/Inserts made to Invoice Matching tables:

- Invc_head
- Invc_non_merch

Function Level Description

Init ()

1. Call retek_init to determine the max_commit_ctr.
2. Get the system date and time using fetchSysdate().
3. Call the function fetchSaSystemOptions() to retrieve the unit_of_work (UOW).

Process ()

1. Process() gets the store_days to be exported using the first driving cursor.
2. For each store_day to be exported, process() checks whether there are errors associated with this store_day (fetchStoreDayErrorCount()). If so, processing for this store_day is skipped and step 2 is repeated for the next store_day.
3. Process() then attempts to obtain a read lock for this store day. If unsuccessful, processing for this store_day is skipped and step 2 is repeated for the next store_day.
4. Call processStoreDay() to process the store day transactions for DSD invoices.
5. Call postInvoices() to create the appropriate invc_head and/or invc_non_merch records.
6. Call processStoreDayTotals() to process the escheatment totals
7. Call markStoreDayExported() – updates sa_export_log, sets status to exported
8. Call release_lock() to release the lock on the store/day.
9. Steps 2-7 are repeated until there are no more store_days.

Final ()

1. Clean up any memory used.
2. Call updateStoreDayExported() to update sa_export_log for the last store day and system code (SYSE_IM).
3. Call retek_close()
4. Call retek_refresh_thread()

processStoreDay()

1. This function uses the second driving cursor to retrieve all transactions associated with direct store delivery invoice transactions. This function should fetch the appropriate data for inserts into invc_head and invc_non_merch tables.
2. Calls PostInvoices() posts the invoices to invc_head and invc_non_merch tables.
3. Calls the library function markTransactionExported() to insert a record into sa_exported.

processStoreDayTotals()

This function will loop through the library function getTotals() for the current store day and SYSE_IM usage type.

The getTotals() function is being modified to include pass back ref_no1, ref_no2 and ref_no3. Ref_no1 will contain the partner_id, ref_no2 the partner_type and ref_no3 the escheat amount.

```
int getTotal(  
    char    *is_store_day_seq_no,  
    const char *is_usage_type,  
    char    *os_total_seq_no,  
    char    *os_total_value,  
    char    *os_rev_no,  
    char    *os_total_type  
    char    *os_total_type_id,  
    char    os_ref_no1,  
    char    os_ref_no2,  
    char    os_ref_no3  
    char    *os_status,  
    long    il_max_counter,  
    long    il_multiplier);
```

1. Call `post_escheat_invoice()` to create the appropriate `invc_head` and/or `invc_non_merch` records.
2. Mark the total exported by calling the library function `markTotalExported()`.

`calcDueDate()`

Select `duedays` from the `terms` table for the terms associated with the supplier or partner on `sa_tran_head.vendor_no`. `Due_date` = invoice date + `duedays`. Invoice dates should be `sa_tran_head.tran_datetime` that was fetched in the first cursor. The terms discount percent should be taken from the `terms` table.

```
SELECT TO_CHAR((TO_DATE(invoice date, 'YYYYMMDDHH24MISS')
+ duedays), 'YYYYMMDD'),
```

```
percent
```

```
FROM terms
```

```
WHERE terms = terms of the supplier or partner
```

`getNewInvcId()`

This function should call the package `INVC_SQL.NEXT_INVC_ID` to retrieve the next invoice id in the sequence. Include standard package error handling around the package call.

`postInvoices()`

Note: Partner, supplier, and date information will all be validated in ReSA so there is no need to validate in the export. ReSA validates to ensure that merchandise invoices are only associated with a supplier, not a partner. Validates partner, date, and numbers as well.

Calls `check_invc_exists()` to check if the invoice exists on the `invc_head` tables. If a record is found, update it, otherwise call `GetNewInvcId()` and insert a new record to `invc_head`. For non-merchandise invoices, be sure to update/insert the `invc_non_merch` record as well. A record should only be written to `invc_non_merch` if the invoice is a non-merchandise invoice. This can be determined by looking at the `sub_tran_type` on `sa_tran_head`. If the `sub_tran_type` is 'EV', then it's a non-merchandise invoice. If it's 'MV', then it's a merchandise invoice. If the invoice does not exist on the `invc_head` table call `post_new_invoice()`. If the invoice already exists on the `invc_head` table call `update_invoice()`.

1. Call `calcDueDate()` to determine what the due date is for each invoice.
2. Call `get_invc_head_data()` to get other columns :
`terms, payment_method, freight_terms, currency_code, exchange_rate, addr_key` for the invoice associated with a supplier or partner.

3. Use the mapping below to determine what values to use when inserting into invc_head and/or invc_non_merch.

Field Mapping between ReSA and Invoice Matching

INVC_HEAD	Required ?	Datatype	Sales Audit Value
INVC_ID	NOT NULL	NUMBER(10)	If the invoice is not matched with an existing one, invc_id will be a system generated number (invc_sql.next_invc_id).
INVC_TYPE	NOT NULL	VARCHAR2(1)	'I' for Merchandise Invoice, 'N' for Non-merchandise Invoice
SUPPLIER		NUMBER(10)	DSD - Sa_tran_head.vendor_no Escheat – NULL
EXT_REF_NO		VARCHAR2(30)	DSD - Sa_tran_head.vendor_invc_no Escheat – 'E' concatenated with State or Partner ID (Which will be state)
STATUS	NOT NULL	VARCHAR2(1)	Code for the status of the invoice. Valid values are U for unmatched, R for partially matched, M for matched, A for approved and P for posted. Invoice statuses are held on the codes table under the code type 'IMST'. DSD - Default to 'U' using IMST_U. Escheat – Default to 'A' using IMST_A
EDI_INVC_IND	NOT NULL	VARCHAR2(1)	'N'
EDI_SENT_IND	NOT NULL	VARCHAR2(1)	'N'
MATCH_FAIL_IND	NOT NULL	VARCHAR2(1)	Indicates whether or not an invoice has failed a match attempt. Valid values are 'Y' or 'N'. Default to 'N' using YSNO_N.
REF_INVC_ID		NUMBER(10)	N/A – Used for types 'C', 'D', 'R'. Default to NULL.

INVC_HEAD	Required ?	Datatype	Sales Audit Value
REF_RTV_ORDER_NO		NUMBER(6)	N/A – Used for types ‘C’, ‘D’, ‘R’. Default to NULL.
REF_PRICE_CHANGE		NUMBER(8)	N/A – Used for types ‘C’, ‘D’, ‘R’. Default to NULL.
REF_RSN_CODE		VARCHAR2(6)	N/A – Used for types ‘C’, ‘D’, ‘R’. Default to NULL.
TERMS		VARCHAR2(15)	Defaulted from sups or partner table if DSD transaction or partner table if escheatment.
DUE_DATE	NOT NULL	DATE	Defaulted based on terms of the supplier or partner.
PAYMENT_METHOD		VARCHAR2(6)	Code identifying the payment method for the invoice, indicating how the invoice will be paid. Valid values include 'LC' for letter of credit, 'WT' for wire transfer, and 'OA' for open account. Other values maybe added by the client as desired. Payment methods will be held on the codes table under a code type of 'PAYM'. Defaulted based on the payment_method of the vendor on the invoice for Merchandise Invoice. Default to NULL for Non merchandise Invoice.
TERMS_DSCNT_PCT		NUMBER(12,4)	Discount that will be applied to the invoice if the invoice is paid by the due date. Default to the terms.pct of the terms associated with the vendor on the invoice.
TERMS_DSCNT_APPL _IND	NOT NULL	VARCHAR2(1)	Indicates whether or not the terms discount has been applied to the total cost of the invoice or not. Valid values are 'Y' or 'N'. Default to ‘N’ using YSNO_N.

INVC_HEAD	Required ?	Datatype	Sales Audit Value
TERMS_DSCNT_APPL _NON_MRCH_IND	NOT NULL	VARCHAR2(1)	This field will indicate if the specified terms discount should be applied to non-merchandise costs.
FREIGHT_TERMS		VARCHAR2(2)	Indicator that references the freight terms associated with the invoice. Default from sups table for merchandise invoice. Non Merchandise Invoice: NULL.
CREATE_ID	NOT NULL	VARCHAR2(30)	'ReSA'
CREATE_DATE	NOT NULL	DATE	The data the invoice was entered in the system – vdate
INVC_DATE	NOT NULL	DATE	Date the invoice was issued by the supplier – sa_tran_head.tran_datetime for existing invoice. For New Invoice default to vdate.
MATCH_ID		VARCHAR2(30)	Oracle user ID of the user that matched the invoice. Default to NULL.
MATCH_DATE		DATE	Date the invoice was matched. Default to NULL.
APPROVAL_ID		VARCHAR2(30)	Oracle user ID of the user that approved the invoice match. Default to NULL.
APPROVAL_DATE		DATE	Date the invoice match was approved. Default to NULL.
FORCE_PAY_IND	NOT NULL	VARCHAR2(1)	Indicates whether or not the invoice is to be force paid (paid before being matched to receipts). Valid values are Y or N. Default to 'N' using YSNO_N.
FORCE_PAY_ID		VARCHAR2(30)	Oracle ID of the user that marked the invoice for force payment. This field will only have a value if the force_pay_ind = 'Y'. Default to NULL.

INVC_HEAD	Required ?	Datatype	Sales Audit Value
POST_DATE		DATE	Date the invoice was posted to the AP staging tables.
CURRENCY_CODE	NOT NULL	VARCHAR2(3)	Code identifying the currency in which the invoice is held. Default the supplier's or partner's currency.
EXCHANGE_RATE		NUMBER(20,10)	Exchange rate at which the invoice is held. Default from currency_rates table based on the currency of the supplier/partner.
TOTAL_MERCH_COST		NUMBER(20,4)	Total merchandise cost for the invoice. This field will be held in the invoice currency. For DSD merchandise invoices, this field should hold the total cost of the invoice (sa_tran_tender.tender_amt). Default to NULL for DSD non-merchandise and escheatment invoices.
TOTAL_QTY		NUMBER(12,4)	Total quantity of items on the invoice. This field is optional, and only needs to be entered if total quantity matching will be performed on the invoice. Quantity will not be captured in ReSA. Default to NULL.
DIRECT_IND	NOT NULL	VARCHAR2(1)	Indicates whether the invoice was created for a direct store delivery order via the Quick Order Entry form in which the invoice was already paid. Valid values are 'Y' -Yes and 'N' -No. Default to 'Y' using YSNO_Y.
PARTNER_TYPE		VARCHAR2(6)	Type of partner assigned to the invoice. This field will always be 'EV' for Expense Vendor. Default using PTAL_EV.

INVC_HEAD	Required ?	Datatype	Sales Audit Value
PARTNER_ID		VARCHAR2(10)	DSD - Sa_tran_head.vendor_no
ADDR_KEY	NOT NULL	NUMBER(6)	Indicates which vendor invoice address should be associated with the invoice. Default to the primary address for the invoice address type on ADDR table (Addr_type = 05 and primary_addr_ind = 'Y').
PAID_IND	NOT NULL	VARCHAR2(1)	Default to 'Y' (YSNO_Y) for all DSD transactions –they will be assumed paid. Set to 'N' (YSNO_N) for Escheatment invoices – they will be assumed not paid .
PAYMENT_REF_NO		VARCHAR2(13)	DSD - Sa_tran_head.payment_ref_no Escheat – NULL
PAYMENT_DATE		DATE	Date that the invoice was paid from the POS system – sa_tran_head.tran_datetime. Escheat – NULL
PROOF_OF_DELIVERY_NO		VARCHAR2(30)	DSD – Sa_tran_head.proof_of_delivery_no Escheat – NULL
CE_ID		NUMBER(10)	NULL
OBLIGATION_KEY		NUMBER(10)	NULL
COMMENTS		VARCHAR2(255)	NULL

INVC_NON_MERCH	Required?	Datatype	Sales Audit Value
INVC_ID	NOT NULL	NUMBER(10)	The invc_id on invc_non_merch should correspond to the one on invc_head.
NON_MERCH_CODE	NOT NULL	VARCHAR2(6)	Code identifying the non-merchandise cost being added to the invoice. These codes will be held on the non_merch_code_head table . For DSD transactions, this field should be set to the reason_code from sa_tran_head. Should be 'E' for Escheatment invoices.
NON_MERCH_AMT		NUMBER(20,4)	Amount of the non-merchandise cost, specified by the non-merchandise code that has been invoiced for. This field will be held in the invoice currency. DSD – sa_tran_tender.tender_amt Escheat – total_value from getTotals()
VAT_CODE		VARCHAR2(6)	Default to NULL.
SERVICE_PERF_IND	NOT NULL	VARCHAR2(1)	Indicates if a service non-merchandise cost has actually been performed. Valid values are 'Y' (service has been performed) or 'N' (service has not been performed or non-merchandise cost is not a service cost). For DSD, if proof of delivery is provided, this field should be 'Y' (YSNO_Y). Should be set to 'N' (YSNO_N) for escheatment invoices.
STORE		NUMBER(4)	Indicates the store at which the service was performed. Should be populated with the DSD invoice store number. Null for escheatment.

Scheduling Considerations

Processing Cycle: Anytime – Sales Audit is a 24/7 system.

Scheduling Diagram: This module should be executed after the ReSA transaction import process.

Threading Scheme: N/A

Locking Strategy

Locking will be performed via the `get_lock` and `release_lock` library functions to lock the store-day during processing.

Restart/Recovery

The logical unit of work for this module is defined as a unique store/day combination. Records will be fetched, updated and inserted based on the `commit_max_ctr` specified on the `restart_control` table. Only two commits will be done, one to establish the store/day lock and another at the end, to release the lock after a store/day has been completely processed.

In case of failure, we rollback all work done to the point right after the call to `get_lock()` and then we release the lock. Thus, we assume that the rollback segment is large enough to hold all inserts into `sa_exported` for one store_day. If this is not the case, we need to increase the size of the rollback segment. The `EXEC SQL SAVEPOINT` statement is used to save the state of the database after the call to `get_lock()`.

Restart recovery is implicit in the program, as only store_days with a `sa_export_log.status` of 'R'eady (SAES_R) will be selected for processing. Since we set this status to 'E'xported (SAES_R) after a store_day is processed, then on restart, store_days that have been processed will be skipped.

Driving cursors:

The program has three driving cursors: one to fetch store_days to be exported, another to fetch invoice matching transactions to be exported for the store_days fetched in the first cursor, and the last to fetch escheatment totals to be exported to Invoice Matching.

The following cursor will be used to retrieve the valid store/day identifiers that must be processed:

```
SELECT sd.store_day_seq_no,
       TO_CHAR(sd.business_date, 'YYYYMMDD'),
       el.seq_no,
       sd.store
FROM sa_store_day sd, sa_export_log el
WHERE sd.store_day_seq_no = el.store_day_seq_no
      AND sd.store_status = :SASS_C    /* Closed */
      AND sd.data_status = :SADS_F    /* Fully Loaded */
      AND sd.audit_status IN (:SAAS_A, :SAAS_S, :SAAS_H) / *Audit Pending,
Store Errors Pending, HQ Errors Pending */
      AND el.system_code = :SYSE_IM
      AND el.status = :SAES_R /* Ready to be exported */
ORDER BY sd.store, sd.business_date;
```

The second driving cursor selects DSD invoice transactions from transaction tables ('PADIOU').

```
SELECT h.tran_seq_no,
       h.rev_no,
       TO_CHAR(h.tran_datetime,'YYYYMMDDHH24MISS'),
       h.tran_no,
       h.tran_type,
       h.sub_tran_type,
       h.reason_code,
       h.vendor_no,
       h.vendor_inv_no,
       h.payment_ref_no,
       h.proof_of_delivery_no,
       h.status,
       t.tender_amt
       :SAFD_P /* Positive Transaction/Total */
FROM sa_tran_head h
WHERE h.store_day_seq_no = :is_store_day_seq_no
      AND h.tran_seq_no = t.tran_seq_no
      AND h.tran_type IN (:TRAT_PAIDOU)
      AND h.status != :SAST_D
      AND h.sub_tran_type IN ('EV','MV')
      AND NOT EXISTS
        (SELECT ex.tran_seq_no
         FROM sa_exported ex
         WHERE ex.tran_seq_no = h.tran_seq_no
              AND ex.store_day_seq_no = :is_store_day_seq_no)
```

```
AND NOT EXISTS    /* and no errors for the transaction. */  
  
    (SELECT er.tran_seq_no  
  
        FROM sa_error er, sa_error_impact ei, sa_tran_head h  
  
        WHERE h.tran_seq_no = er.tran_seq_no  
  
            AND er.error_code = ei.error_code  
  
            AND ei.system_code = :SYSE_IM  
  
            AND er.hq_override_ind != :YSNO_Y))  
  
ORDER BY h.tran_datetime;
```

The last driving cursor is embedded in the getTotals() function. This function is called with a usage type of SYSE_IM. For each escheatment transaction that is processed, write to the invc_head and invc_non_march tables.

Design Assumptions

All DSD invoices will be assumed paid.

All Escheatment invoices will be assumed not paid.

[saexpsfm.pc]

Design Overview

This module will post Store/day fuel transactions and totals to the Oracle Site Fuel Management (SFM) application. There are four different types of fuel transactions that will be transferred to SFM. The 'Pump test' (tran_type: 'PUMPT') transactions will be summarized in a total of usage type SFMP and posted to the SFM metered sales API call (eas_metrd_sales_pub.create_meter_sales_volume). The 'fuel sales' ('SALES') transactions (for fuel items) will be summarized and posted to the SFM metered sales API call (eas_metrd_sales_pub.create_meter_sales_volume). The 'Meter Reading' ('METER') transactions will be posted to the SFM meter reading API call (eas_meter_readg_pub.create_meter_reading), and the 'Tank dip' ('TANKDP') transactions will be posted to the SFM tank dip API call (eas_tank_dip_pub.create_tank_dip).

This batch module will require two input parameters in addition to userid/password. The first is used to determine which store will be processed during the execution of the program.

The second will be a 0 or a 1 to determine whether multiple errors from the SFM API should be reported. A 0 means only the first error should be reported and a 1 means up to 10 errors should be reported.

Table	Operations Performed			
	Select	Insert	Update	Delete
Period	Yes	No	No	No
Sa_system_options	Yes	No	No	No
Sa_store_day	Yes	No	No	No
Sa_store_day_read_lock	Yes	Yes	No	Yes
Sa_export_log	Yes	No	Yes	No
Sa_total_value	Yes	No	No	No
Sa_pos_value	Yes	No	No	No
Sa_store_value	Yes	No	No	No
Sa_sys_value	Yes	No	No	No
Sa_hq_value	Yes	No	No	No
Sa_tran_head	Yes	No	No	No
Sa_tran_item	Yes	No	No	No
Sa_exported	No	Yes	No	No
Sa_total_usage	Yes	No	No	No
Sa_error	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Anytime – Sales Audit 9.0 is a 24/7 system.

Scheduling Diagram: This module should be executed after the ReSA transaction import process, and initiated by a specific store.

Threading Scheme: N/A

Restart Recovery

The logical unit of work for this module is defined as a unique store/day combination. Records will be fetched, updated and inserted in batches of commit_max_ctr.

Restart recovery is implicit in the program as only store_days with a sa_export_log.status of 'R'eady will be selected for processing. Since we set this status to 'E'xported after a store_day is processed, then on restart, store_days that have been processed will be skipped.

Store_days to be processed are locked with a Read lock before processing. A commit is performed by the function in order to obtain this lock.

The function updateStoreDayExported also does a commit to mark sa_export_log.status to 'L', while the function markStoreDayExported does a commit if commit_max_ctr store_days are to be marked to 'L'.

The SFM API performs an “autonomous commit” which means that their commit does not affect our data.

Driving cursors:

The program has three driving cursors: one to fetch store_days to be exported, another to fetch fuel transactions to be exported for the aforementioned store_days and the last to fetch all sales totals to be exported to SFM.

The libresa library function fetchStoreDayToBeExported cannot be used for this module since only store_days for a specific store number may be processed in a given instance of the program. Therefore, the following cursor will be used to retrieve the valid store/day identifiers that must be processed:

```
SELECT sd.store_day_seq_no,
       TO_CHAR(sd.business_date, 'YYYYMMDD'),
       el.seq_no,
       s.currency_code,
       sd.data_status
FROM sa_store_day sd, sa_export_log el, store s
WHERE sd.store = :il_store
      AND sd.store_day_seq_no = el.store_day_seq_no
      AND sd.store_status = :SASS_W
      AND sd.data_status IN (:SADS_F, :SADS_P)
      AND sd.audit_status IN (:SAAS_A, :SAAS_S, :SAAS_H, :SAAS_T)
      AND el.system_code = :SYSE_SFM
      AND el.status = :SAES_R
      AND s.store = :il_store
      AND el.seq_no = (SELECT max(seq_no)
                      FROM sa_export_log
                      WHERE store_day_seq_no = sd.store_day_seq_no
                        AND system_code = :SYSE_SFM)
ORDER BY sd.business_date;
```

This cursor only fetches store_days for the given store which are 'R'eady for export.

The second driving cursor selects pump test, tank dip and meter reading transactions from transaction tables ('PUMPT', 'TANKDP', 'METER' transaction type respectively). The proper SFM API function is then called, depending on the transaction type

```
SELECT h.tran_seq_no,
       h.rev_no,
       TO_CHAR(h.tran_datetime,'YYYYMMDDHH24MISS'),
       h.tran_no,
       h.tran_type,
       h.sub_tran_type,
       h.ref_no1,
       h.ref_no2,
       h.ref_no3,
       h.ref_no4,
       i.ref_no5,
       i.ref_no6,
       i.ref_no7,
       i.ref_no8,
       h.status,
       i.sku,
       i.qty,
       i.unit_retail,
       i.pump
FROM sa_tran_head h, sa_tran_item i
WHERE h.store_day_seq_no = :is_store_day_seq_no
      AND h.tran_seq_no = i.tran_seq_no
      AND h.tran_type IN (:TRAT_TANKDP, :TRAT_METER)
      AND i.dept = :is_fuel_dept
      AND h.status != :SAST_D
      AND NOT EXISTS
```

```
(SELECT ex.tran_seq_no
FROM sa_exported ex
WHERE ex.tran_seq_no = h.tran_seq_no
AND ex.store_day_seq_no = :is_store_day_seq_no)
ORDER BY h.tran_datetime;
```

The last driving cursor is embedded in the function getTotalLike. This function is called with a usage type of 'SFM%', that is it will return all totals that have a usage type beginning with the string 'SFM'. We expect a usage type of 'SFMS' for sales totals and 'SFMD' for non-sales totals, also known as drive off totals. The SFM API function eas_metrdr_sales_pub.create_meter_sales_volume is called for each total retrieved.

Program Flow

Although this module is designed for an individual store, the flow of data is essentially the same as the standard ReSA export design.

Shared Modules

librettek library functions:

- retek_init/retek_close - to initialize and close Retek's restart/recovery module
- fetchSysdate – to get the current date and time
- fetchSaSystemOptions – to get ReSA system options

libresa library functions:

- fetchStoreDayErrorCount – to determine whether there are errors for the current store_day
- markStoreDayExported – to mark a store day as updated
- updateStoreDayExported – to force commit of status 'E' on sa_export_log for store days that have been exported
- getTotalLike – to fetch totals that match 'SFM%'
- get_lock – to establish a read lock on a store_day
- release_lock – to release a store_day lock

SFM API functions – these functions are called with the data that needs to be exported.

Function Level Description

Init ()

1. Get the system date and time using fetchSysdate().
2. Call the function fetchSaSystemOptions() to retrieve the unit-of-work (UOW) and the fuel department code.

Process ()

1. Process() gets the store_days to be exported using the first driving cursor.
2. For each store_day to be exported, if the UOW on the sa_system_options table is set to 'S'tore_day, process() checks whether there are errors associated with this store_day. If so, processing for this store_day is skipped and step 2 is repeated for the next store_day.
3. Process() then attempts to obtain a read lock for this store day. If unsuccessful, processing for this store_day is skipped and step 2 is repeated for the next store_day.
4. Process then calls processSFMFuelItemCursor().
5. Steps 2-4 are repeated until there are no more store_days.

Final ()

1. Clean up any memory used.
2. Call updateStoreDayExported() to commit changes to sa_export_log.

processSFMFuelItemCursor()

1. This function uses the second driving cursor to get all fuel transactions to be exported to SFM. It first calls the function getComments() to retrieve comments (if any) associated with the transaction.
2. Then ,given a transaction type, it calls the appropriate SFM API. Calls to the latter are made through a “wrapper” function, that is, we call a function which will then call the SFM API (see table). This is done to encompass preparation of data, validation, and error handling into one logical unit.

Transaction Type	Wrapper function
PUMPT	postSFMPumpTest
TANKDP	postSFMTankDip
METER	postSFMMeterReading

Then postSFMSales() is called to post sales and non-sales information.

getComments()

This function retrieves the comment(s) associated with a given transaction and, if multiple comments exist for the same transaction, it concatenates them together up to a length equal to LEN_COMMENTS.

Wrapper functions postSFMTankDip(), and postSFMMeterReading()

1. These are the wrapper functions for the SFM API. They should compute the parameters for the SFM API functions, call them and perform any error handling. Also, some require preparation of data before calling the API. For example, one TANKDP transaction may be on two records on the sa_tran_item table. Thus, both records have to be put into one structure before the API is called.
2. When the API returns an error ('E'xpected or 'U'nexpected), the functions should print either the first error on the error list or up to 10 errors on the error list, depending on the command line argument. None of these errors are fatal from the standpoint of this program. Thus it merely writes the error to the log and continues.
3. Furthermore, these functions perform only minimal validation. For a TANKDP transaction, the tank_dip_closing_volume, tank_dip_height_major and tank_dip_height_minor variables are populated if ref_no1 is "FUEL", and the water_closing_volume, water_height_major and water_height_minor variables are populated if ref_no1 is "WATER". For a METER transaction, ref_no1 denotes the type of the reading taken and is expected to be "C" for closing, "S" for shift change, "P" for price change and "A" for an adjustment. No validation is performed for these transactions.

postSFMSales()

1. In a sense this is a wrapper function too. It calls getTotalLike to get all SFM totals. Only 'P'ositive transactions are posted, that is revisions are not posted. The SFM API function used is eas_metrdr_sales_pub.create_meter_sales_volume.
2. For each total retrieved, the function adds it to a linked list of totals. Each node represents one unique transaction date time/pump/sku/price rollup.
3. Since SFM expects a sales value for all nodes, we put a zero in the field, when no sales value exists for a given node.
4. If it is of usage type SFMS, then the sold_value field is populated, else if it is of usage type SFMD, then the non_sales_value field is populated, else if the usage type is SFMP, then the pump_test_value field is populated.
5. Totals with matching the same ref_no fields are copied into the same node in the linked list.
6. The ref_nos are set up such that: ref_no1 contains the trading_period_datetime in 'YYYYMMDDHH24MISS' format; ref_no2 contains the pump and sku concatenated together with a ":" in between. ref_no3 contains the unit_price. See "Function Design -- SFM Totals" for more information.
7. After all totals have been retrieved, the SFM API is called for each node.

I/O Specification

The "Oracle SFM API" document and the "Interface File – SA to SFM" document outline all required calls and specifics to the SFM application.

[saexpuar.pc]

Design Overview

This module will post specified totals to the *Driscoll UAR* application. Using the typical export process, this module will loop through all available store day combinations. For each store day, all specified totals will be posted to their appropriate output files. All driving cursors will be handled by the *libresa* library functions.

Table	Operations Performed			
	Select	Insert	Update	Delete
Period	Yes	No	No	No
Sa_store_day	Yes	No	No	No
Sa_export_log	Yes	No	Yes	No
Sa_exported	No	Yes	No	No
Sa_exported_rev	Yes	No	No	No
Sa_total_head	Yes	No	No	No
Sa_total	Yes	No	No	No
Sa_bank_store	Yes	No	No	No
Sa_store_day_read_lock	Yes	Yes	No	Yes
Sa_store_value	Yes	No	No	No
Sa_sys_value	Yes	No	No	No
Sa_pos_value	Yes	No	No	No
Sa_hq_value	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Anytime – Sales Audit 9.0 is a 24/7 system.

Scheduling Diagram: This module should be run after the ReSA Totaling process.

Threading Scheme: N/A

Restart Recovery

The logical unit of work for this module is defined as a unique store/day combination. Records will be fetched, updated and inserted in batches of `pl_commit_max_ctr`. Only two commits will be done. One to establish the store/day lock (this will be done by the package) and one at the end after a store/day has been completely processed.

Driving cursor 1:

The `libresa` library function `fetchStoreDayToBeExportedLike` will drive the stores to be processed for any usage type starting with 'UAR'.

Driving Cursor 2:

The `libresa` library function `getTotalLike` will drive the totals to be exported for any usage type starting with 'UAR'.

Shared Modules

`libresa` library functions:

- `fetchStoreDayToBeExportedLike`
- `fetchSaSystemOptions`
- `fetchSysdate`
- `fetchStoredayErrorCount`
- `markStoreDayExported`
- `updateStoreDayExported`
- `markTotalExported`
- `getTotalLike`
- `get_lock`
- `OraNumInit`
- `OraNumAdd`

Function Level Description

init ()

1. Call OraNumInit to initialize string numbers arithmetic operations.
2. Get the current system date from the library function fetchSysdate.
3. Get the unit-of-work by calling the library function fetchSaSystemOptions.

process ()

1. Loop through the libresa library function fetchStoreDayToBeExportedLike.
2. Attempt to obtain a read lock on the store/day with a call to get_lock. If this fails, go on to the next store/day and log the problem to the error log.
3. Call the function processStoreDay for the current store day.
4. Call the function markStoreDayExported.
5. Call the function retek_force_commit.
6. Loop from beginning until the return result of the function fetchStoreDayToBeExportedLike = 1.

final ()

1. Call the library function updateStoreDayExported to write any unwritten store days to the database.
2. Close output files.
3. Clean up any memory used.
4. Call the function retek_close.

addNewOutputFile (char is_usage_type,

char is_business_date,

char is_sysdate) returns integer

This function will generate a new output file for any new usage types retrieved from the getTotalLike function call. It will also add a new file item to a collection of any files currently being written to.

The file collection should contain the following items:

- Usage type – the usage type returned by getTotalLike.

- File name - <usage type>_<business date>_<system date>
- File pointer – Pointer to the output file.
- Wrote header – file header written indicator (0 – no, 1 – yes)
- File sum – ongoing sum of each transaction in file.

getOutputFilePointer (char is_usage_type) returns integer

This function will retrieve the output file pointer for the usage type passed by checking to see if the usage type exists on the output file collection.

If the usage type exists on the file collection, the item number for the collection is returned.

If the usage type does not exist, the function returns -1.

writeStoreDayDetail (FILE *if_file_pointer,
char is_total_id,
char is_store,
char is_business_date,
char is_amount,
char is_total_seq_no,
char is_UAR_tran_code) returns integer

This function will write the current UAR total to the output file specified. Each field is separated by commas and surrounded by double quotes.

UAR Detail record:

Field	Description	Sales Audit value
1	Detail flag	1
2	Serial number	Store number
3	Amount	Total value
4	Transaction date	Transaction date
5	Transaction code	Mapped value: see the function <code>getAdditionalInfo</code> for detailed explanation.
6	User defined value 1	Total sequence number
7	User defined value 2	Nothing
8	User defined value 3	Nothing
9	User defined value 4	Nothing
10	User defined value 5	Nothing
11	User defined value 6	Nothing
12	User defined value 7	Nothing
13	User defined value 8	Nothing
14	User defined value 9	Nothing
15	User defined value 10	Nothing
16	State	Nothing
17	Account	Total identifier
18	End of line	\n

All 18 fields should be concatenated together.

`writeStoreDayHeader (FILE *if_file_pointer,`

`char is_total_id,`

`char is_business_date)` returns integer

This function will write the header record for the current store day to the output file. Each field is separated by commas and surrounded by double quotes.

UAR Header record:

Field	Description	Sales Audit value
1	Header flag	0
2	Account number	Total identifier
3	Source	D
4	Transaction date	Transaction date
5	Organization number	Nothing
6	Format	UAR34
7	End of line	\n

All 7 fields should be concatenated together.

writeStoreDayFooter (FILE *if_file_pointer,

char is_amount) returns integer

This function will write the footer record for the current store day to the output file. Each field is separated by commas and surrounded by double quotes.

UAR Footer record:

Field	Description	Sales Audit value
1	Footer flag	9
2	Beginning balance	+00000000000000
3	Ending balance	“+” the ongoing sum for the file.
4	Available balance	Nothing
5	End of line	\n

All 5 fields should be concatenated together.

CloseOutputFiles () returns integer

This function will loop through the output file collection and call the ‘fclose’ C function for each.

getOutputFileName (char is_usage_type,

char is_business_date,

char is_sysdate,

char os_filename) returns integer

This function will generate the unique file name for the total usage type passed. The filename will have the following structure:

is_usage_type || “_” || is_business_date || “_” || is_sysdate

getAdditionalInfo (char is_total_seq_no,
char is_ref_no1,
char os_total_id,
char os_UAR_tran_code) returns integer

This function retrieves both the total identifier and UAR transaction code for the current total sequence number. The UAR transaction code is retrieved as follows:

If ref_no1 is not null

If ref_no1 maps to the SA_CONSTANTS table (ref_no1 =
SA_CONSTANTS.CONSTANT_ID).

UAR transaction code = SA_CONSTANTS.CONSTANT_VALUE

If ref_no1 does not map to the SA_CONSTANTS table (ref_no1 !=
SA_CONSTANTS.CONSTANT_ID).

UAR transaction code = ref_no1

If ref_no1 is null

UAR transaction code = total identifier

processStoreDay (char is_store_day,
char is_sysdate,
char is_business_date,
char is_store) returns integer

This function will process an entire store days UAR totals.

Loop through all UAR totals by calling the function getTotalLike until the function returns anything but zero.

Call the function getAdditionalInfo.

Determine if the output file exists by calling the function getOutputFilePointer.

If the pointer does not exist, call the function addNewOutputFile to create the new file.

Check to see if the header detail record has already been written for the current file by checking the current item on the output file collection.

If the head has not been written, call the function `writeStoreDayHeader`. Set the header indicator to 1 in the output file collection for the current item.

Write the current total to the current output file by calling the function `writeStoreDayDetail`.

Add the current total value to the running total sum in the output file collection for the current item.

Call the function `markTotalExported`.

I/O Specification

The UAR output file specifications are listed in this document by the functions that write the output:

- `writeStoreDayHeader`
- `writeStoreDayDetail`
- `writeStoreDayFooter`

Technical Issues

None.

[saimpadj.pc]

Design Overview

This module will post external system adjustments to the Sales Audit total value table. The module will be passed an input file name. The input files will be of a standard format as detailed in the “Interface File – SA Import Adj.doc” document.

Table	Operations Performed			
	Select	Insert	Update	Delete
Period	Yes	No	No	No
Sa_store_day	Yes	No	Yes	No
Sa_export_log	Yes	Yes	No	No
Sa_total	Yes	No	No	No
System_options	Yes	No	No	No
Sa_total_usage	Yes	No	No	No
Sa_hq_value	No	Yes	No	No

Scheduling Constraints

Processing Cycle: Anytime – Sales Audit 9.0 is a 24/7 system.

Scheduling Diagram: This module should be executed after the ReSA transaction import process, and before the ReSA totaling process.

Threading Scheme: N/A

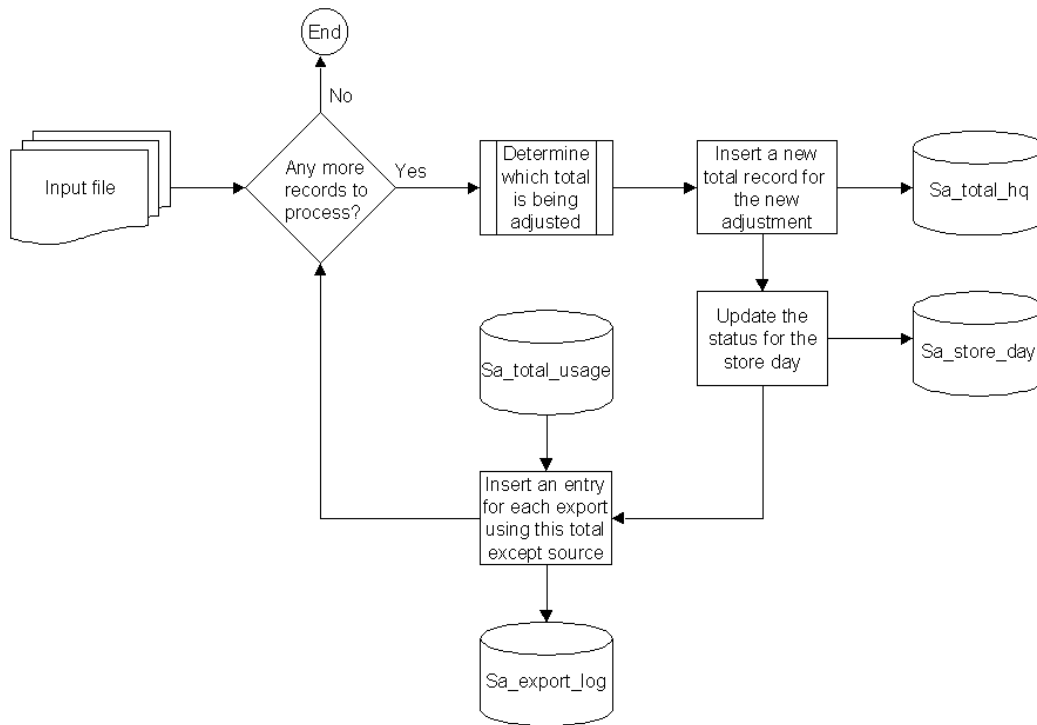
Restart Recovery

The logical unit of work for this module will be a parameterized number of records from the input file. Records will be processed until an internal counter has been reached and a commit will be executed. The Retek library (rettek_2.h) function rettek_commit will be called after each record processed. This function will keep track of the internal counter and commit the work once it has been reached.

Driving cursor:

A standard while loop will read through all records in the input file. All file processing will use the new Retek library (rettek_2.h) functions: rettek_init, rettek_get_record, rettek_write_rej, rettek_commit, rettek_close.

Program Flow



The module Determine which total is being adjusted (as shown above) will depend on the source of the record being processed. If the record is from the UAR application the `uar_source` field will determine which total is being used. If the record is from the General ledger application the `gl_account` field will help determine which total is being adjusted.

The current status of the program passes through the “determine which total is being adjusted” section and just gives the program the total via the input file. The UAR and GL application information is still in question and will be modified when the best procedure is known.

Function Level Description

Init ()

Read in the header from the input file.

Process ()

Loop through the input file adjustment records by calling the Retek library function `retex_get_record`.

Fetch the `store_day_seq_no` from the `sa_total` table.

Get a lock for the record by calling `get_Lock`.

Call the function processAdjustment. If processAdjustment returns a value $\neq 0$ then call the Retek library function retek_write_rej in order to document unidentifiable adjustment records.

Call the Retek library function retek_commit in order to commit on an internally defined number of records processed.

If lock could not be obtained, write an error to the error log.

Final ()

Call retek_close

Clean up any memory used.

processAdjustment (Char il_store_day_seq_no)

This function will complete one total adjustment by using the following processes:

Call the function determineTotal in order to identify the total identifier for the incoming adjustment.

Call the function postAdjustmentTotal in order to move the new adjustment amount into the current total table.

Call the function updateStoreDayStatus in order to ensure that the store day is re-calculated by the totaling process.

Call the function postExportLog in order to insert new records for all export requests except the source application of the adjustment.

Return 0 if successful adjustment posting.

Return 1 if unable to identify total.

Return -1 if a database error occurs.

determineTotal (o_totalIdentifier *char) as Long

(if necessary – still waiting on) This function will retrieve the total identifier for the current adjustment. This function will be the only location that will be checking specific source application data; any changes to the way applications handle exporting total adjustments will only have to be modified in this function. The following procedures will occur:

If the source of the current adjustment record is the General Ledger.

Determine the total identifier by tracing the store, date and GL account number back to the GL mapping table (depending on which financial application is being used).

If the source of the current adjustment record is the UAR.

Determine the total identifier by tracing the store, date and UAR source type back to the total definition table.

If the source of the current adjustment record is not UAR or General Ledger.

Return the Total field in the o_totalIdentifier argument.

Return 0 if total identifier is found.

Return 1 if total identifier is not found.

Return -1 if a database error occurs.

postAdjustmentTotal () as Long

This function will insert the new total value record with the new data from the adjustment record. The following procedures will occur:

Fetch the max value_rev_no from the four value tables.

Find the max of the values returned

Insert the new total value into sa_hq_value table.

Return 0 if successful.

Return 1 if the total was not found in the total value table.

Return -1 if a database error occurred.

updateStoreDayStatus () as Long

This function will update the status field in the sa_store_day table. The following procedures will occur:

Update the sa_store_day table setting the audit status field to 'Re-Totaling required' ('R') and the audit changed date to the sysdate.

Return 0 if successful.

Return 1 if the store day record was not found in the sa_store_day table.

Return -1 if a database error occurred.

postExportLog () as Long

This function will make an entry in the export log (sa_export_log) for every application that requires export data for the current total being adjusted except the source application of the adjustment. The following procedures will occur:

Open a cursor to retrieve all export applications for the current total identifier:

Select usage_type

From sa_total_usage

Where total = :ls_total

And usage_type != :ls_data_source

Fetch the max seq_no from the sa_export_log table.

In a standard while loop

Insert a record into the export log (sa_export_log) for each usage type returned from the above query. Place the usage_type into the system_type field, and enter the status as 'R' for ready to export.

Return 0 if successful.

Return -1 if a database error occurred.

I/O Specification

The standard adjustment input format file can be found in the "Interface File – SA Import Adj.doc" document.

Stock Ledger Append [salapnd]

Design Overview

The sole purpose of this program is to move data from the transaction staging table into the historical transaction table. This requires placing a lock on the staging table to ensure that no new data will be added to it while the movement is occurring (to handle trickling or real-time processing), moving the data to the historical table, and finally truncating the data from the staging table.

Scheduling Constraints

Processing Cycle: Should occur after all extractions have completed (RMS – saldly, RDW etc.)

Scheduling Diagram: N/A

Pre-Processing: salstage, all extraction, and all processing

Post-Processing: N/A

Threading Scheme: Threading will be implicit via the use of the Oracle Parallel Query Option. The insert/select query should be tuned for each specific environment to achieve the best throughput.

Restart Recovery

No specific restart/recovery scheme needs to be defined because of the limited scope of the module. However, in cases where the tran_data table is very large, a particularly large rollback segment may be specified to reduce the risk of exceeding rollback segment space. This will depend on the size of normal rollback segments and the size of the tran_data table.

Program Flow

N/A

Shared Modules

N/A

Function Level Description

Init():

The rollback segment should be specified to a large enough size so that the entire rollup will complete in a single transaction.

Process()

This function first calls the lock_table function to lock the if_tran_data table. It then calls the append_history function to append the data from if_tran_data on to the tran_data_history table. Finally, it calls lock table to release the lock on the tran_data table.

Lock_table()

This function places or remove a lock on a table passed in. It will place or release the lock depending on a input parameter.

Append_history()

This function appends the data on if_tran_data on to the tran_data_history table.. This is done using an insert/select statement with a hint for parallelism. The degree of parallelism should be customized to each run-time environment.

I/O Specification

N/A

Technical Issues

N/A

Daily Stock Ledger Processing [saldly]

Design Overview

First the program reads in the calendar type option (regular or 454), then gets last day of the month for the previous and current month. If a month has been closed (by running 'salmth'), no further transactions for that month can be processed. If any are found when this program runs, they are not processed and a message is logged indicating the problem. The transactions will be left on TRAN_DATA after the run. A decision must be made where to put the data. If it is decided to put this data in the current month, the transaction date should be changed accordingly, and the transaction will then be processed during the next run of this program.

The main process of this program reads TRAN_DATA with a processed_ind of 'N' and transaction date greater than the last end-of-month date and the appropriate field on DAILY_DATA is updated based on the transaction type. Rows on DAILY_DATA are inserted if does not exist.

After the tran data has been processed the processed_ind field on tran data is updated to 'Y' which signifies that it has been processed.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DEPT	No	Yes	No	No	No
SUBCLASS	Yes	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No
TRAN_DATA	No	Yes	No	Yes	No
DAILY_DATA	Yes	No	Yes	Yes	No

Scheduling Constraints

Processing Cycle: PHASE 3 (daily)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: STORE_WH
V_restart_store_wh

Restart Recovery

```
SELECT sku,
       dept,
       class,
       subclass,
       store,
       wh,
       TO_CHAR(tran_date),
       TO_NUMBER(TO_CHAR(tran_date, 'DD')),
       TO_NUMBER(TO_CHAR(tran_date, 'MM')),
       TO_NUMBER(TO_CHAR(tran_date, 'YYYY')),
       tran_code,
       SUM(NVL(units,0)),
       SUM(NVL(total_cost,0)),
       SUM(NVL(total_retail,0)),
       ';' || TO_CHAR(tran_data.sku) ||
       ';' || TO_CHAR(tran_data.store)
FROM tran_data,
     v_restart_store_wh rv
WHERE tran_date > TO_DATE(:last_eom_date, 'DDMMYYYY')
AND processed_ind = 'N'
AND rv.driver_value   = tran_data.store
AND rv.driver_name    = :ora_restart_driver_name
AND rv.num_threads    = :ora_restart_num_threads
AND rv.thread_val     = :ora_restart_thread_val
AND (tran_data.sku > NVL(:ora_restart_sku, tran_data.sku - 1))
```



```
OR (tran_data.sku = :ora_restart_sku AND
    (tran_data.store > :ora_restart_store)))

GROUP BY sku,
    dept,
    class,
    subclass,
    store,
    wh,
    tran_date,
    tran_code

ORDER BY sku,
    store;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

End of Half Stock Ledger Processing [saleoh]

Design Overview

This program deletes rows from MONTH_DATA, DAILY_DATA, WEEK_DATA, HALF_DATA, MONTH_DATA_BUDGET, and HALF_DATA_BUDGET which are eighteen months or older.

For each subclass/location, this program will insert one row into half_data for the next half. For each department/location, it inserts six rows (one for each month) into month_data_budget and one row into half_data_budget for the same half of next year. The shrinkage percent on half_data_budget for the same half of next year will be calculated from half_data of the same half of current year. For example, if this program runs at the end of the first half of 1997 (i.e. half_no = 19971) :

- Insert one row on half_data for half_no = 19972 for all subclass/location combinations
- Insert six rows on month_data_budget for half_no = 19981 for all department/location combinations
- Insert one row on half_data_budget for half_no = 19981 for all department/location combinations
- For each department/location, sum up shrinkage amount and sales amount from all the subclass/location records on half_data with subclass belongs to this department and half_no = 19971 and calculates a shrinkage percent to update half_data_budget.shrinkage_pct for this department/location and half_no = 19981.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DAILY_DATA	Yes	No	No	No	Yes
HALF_DATA	No	Yes	Yes	No	Yes
HALF_DATA_BUDGET	No	Yes	Yes	No	Yes
MONTH_DATA	Yes	No	No	No	Yes
MONTH_DATA_BUDGET	No	Yes	Yes	No	Yes
SYSTEM_VARIABLES	No	Yes	No	Yes	No
SYSTEM_OPTIONS	No	Yes	No	No	No
WEEK_DATA	No	No	No	No	Yes

Scheduling Constraints

Processing Cycle:	PHASE 3 (end-of-half)
Scheduling Diagram:	Must run after completion of the salmth program for month 6 of the current half, and before running the salmth program for month 1 of next half.
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

This program first checks to make sure that salmth for month 6 of current half has been completed.

- purges rows on daily_data, week_data, month_data, half_data, month_data_budget and half_data_budget that are 18 months or older
- inserts 1 row on half_data for next half for all subclass/location combinations
- inserts 6 rows on month_data_budget for next year for all dept/location combinations
- inserts 1 row on half_data_budget for next year all dept/location combinations and rolls-up this year's HALF_DATA.shrinkage_pct at subclass level to HALF_DATA_BUDGET.shrinkage_pct at dept level for next year to create the budget shrinkage percent base for next year. To do this,
- first sums up inter_stocktake_shrink_amt and inter_stocktake_sales_amt on half_data from subclass to dept level

- re-calculate shrinkage_pct at the dept level =
- $(\text{sum inter\_stocktake\_shrink\_amt} / \text{sum inter\_stocktake\_sales\_amt}) * 100$
- update half_data_budget.shrinkage_pct for next year using the above calculated shrinkage_pct

I/O Specification

N/A

Technical Issues

N/A

Stock Ledger & Budget Tables Insert [salins]

Design Overview

This program reads the table stock_ledger_inserts and inserts rows into month_data, week_data, half_data, month_data_budget, and half_data_budget for new locations added to the system since the last run. It also inserts rows into month_data, week_data and half_data for new subclasses and into month_data_budget and half_data_budget for new departments.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
STOCK_LEDGER_INSERTS	No	Yes	No	No	Yes
WEEK_DATA	Yes	No	Yes	No	No
MONTH_DATA	Yes	No	Yes	No	No
HALF_DATA	Yes	No	Yes	No	No
MONTH_DATA_BUDGET	Yes	No	Yes	No	No
HALF_DATA_BUDGET	Yes	No	Yes	No	No

Scheduling Constraints

Processing Cycle: PHASE 0 (daily)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

```
select type_code,
       nvl(dept,-1),
       nvl(class,-1),
       nvl(subclass,-1),
       nvl(store,-1),
       nvl(wh,-1)
from stock_ledger_inserts
order by type_code desc;
```

Program Flow

N/A

Shared Modules

STKLEDGR_INSERTS_SQL.ADD_LOC: A call is made to this package function when a new store or warehouse is added to the system. This creates the week_data, month_data and half_data records at the location / subclass level.

STKLEDGR_INSERTS_SQL.ADD_SUBCLASS: A call is made to this package function when a new subclass is added to the system. This creates the week_data, month_data and half_data records at the location / subclass level.

STKLEDGR_INSERTS_SQL.ADD_DEPT_BUDGET: A call is made to this package function for each new department or location that is added to the system. Budget numbers are held at a department / location level and this creates the month_data_budget and half_data_budget records at the location / department level.

Function Level Description

This program reads the table stock_ledger_inserts and inserts rows into week_data, month_data, half_data, half_data_budget and month_data_budget.

Detailed description of the functions of this program are as follows :

Set a cursor on stock_ledger_inserts to read one row at a time :

if type_code = 'S'

insert one row of month_data for the month to be processed in the next salmth run

(ie. the half_no/month_no for (system_variables.last_eom_date + 1))

for this store and all subclass combinations

if system_options.stock_ledger_time_level_code = 'W'

insert week_data for each of the weeks, starting from the first week of

the month to be processed in the next salmth run, until the last week processed in the last salweek run for this store and all subclass combinations

end if

(example : if we are in the second week of June, and salmth has not run for May yet, we would insert month_data for month of May, and week_data for all the weeks in May plus 1st week in June)

insert one row of half_data for the half with half_no =
system_variables.last_eom_next_half_no for this store and all subclass combinations

insert 6 rows of month_data_budget for each of the current half and next half for this

store and all dept (except dept 9999) combinations (identical to inserting month_data in mth_data_add_store.sql)

insert 1 row each of half_data_budget for the current half and next half for this store

and all dept (except dept 9999) combinations

else

if type_code = 'W'

insert one row of month_data for the month to be processed in the next salmth run

for this store and all subclass combinations (ie. the half_no/month_no for
(system_variables.last_eom_date + 1))

if system_options.stock_ledger_time_level_code = 'W'

insert week_data for each of the weeks, starting from the
first week of

the month to be processed in the next salmth run, until the last week processed in
the last salwka run for this warehouse and all subclass combinations

end if

insert one row of half_data for the half with half_no =

system_variables.last_eom_next_half_no for this warehouse and all subclass
combinations

insert 6 rows of month_data_budget for each of the current half and next
half for this

warehouse and all dept (except dept 9999) combinations (identical to inserting
month_data in mth_data_add_wh.sql)

insert 1 row each of half_data_budget for the current half and next half
for this

warehouse and all dept (except dept 9999) combinations

else

if type_code = 'B'

insert one row of month_data for the month to be processed in the next
salmth run

for this subclass and all location combinations (ie. the half_no/month_no for
(system_variables.last_eom_date + 1))

if system_options.stock_ledger_time_level_code = 'W'

insert week_data for each of the weeks, starting from the first week of the month to be processed in the next salmth run, until the last week processed in the last salwka run for this subclass and all locations combinations

end if

insert one row of half_data for the half with half_no = system_variables.last_eom_next_half_no for this subclass and all location combinations

else

if type_code = 'D'

insert 6 rows of month_data_budget for the current half and 6 for next half for this dept and all location combinations (identical to inserting month_data in mth_data_add_dept.sql)

insert 1 row of half_data_budget for the current half and one for next half for this dept and all location combinations

Delete all rows on stock_ledger_inserts after all rows have been processed

I/O Specification

N/A

Technical Issues

N/A

Monthly Stock Ledger Processing [salmth]

Design Overview

The purpose of this program is to sum up the monthly transaction totals from DAILY_DATA and calculate the closing stock and gross margin for the current month on MONTH_DATA. The procedure varies depending on the following factors:

- Whether the retail or cost method of accounting is used. Depending on the setting of DEPS.profit_calc_type -- 1 = cost, 2 = retail
- Whether a stock count of Unit & Dollar type has occurred during the month -
- Determined by the presence or absence of a STAKE_PROD_LOC row by dept/class/subclass/store/wh.

Certain checks are made to ensure that the program is being run at an appropriate point in time.

1. The current date (period.vdate) must not be earlier than the next due eom_date (SYSTEM_VARIABLES.next_eom_date)
2. If a stocktake has been done during the month, all stocktake results must have been processed.

Once the timing is verified each subclass/location record on month_data is processed for the current month. For each record fetched, profit calculation type and purchase type are retrieved from deps table, and budgeted shrinkage percent are retrieved from half_data_budget table.

If a stock count occurs during the current month, stkdly.pc would have already updated the stock count's book stock and actual stock fields on month_data (i.e. stocktake_bookstk_cost (& retail) and stocktake_actstk_cost (& retail) on month_data). The difference between the book stock and actual stock will be used by this program to adjust the closing stock value for the current month.

In addition, this program calculates a shrinkage amount as follows :

If budget shrinkage indicator = 'Y' :

shrinkage amount = budgeted shrinkage percent * sales amount for the month

else

shrinkage amount = - (stock_adj_cost or retail)

depending on cost or retail method is used

If stock count did not occur during the month, the above calculated shrinkage amount will be used to reduce the closing stock for this month.

At the same time, this program adds the above calculated shrinkage amount and sales amount for this month into `inter_stocktake_shrink_amt` and `inter_stocktake_sales_amt` fields on `month_data`, which have been accumulated since the last stock count.

Else

`inter_stocktake_shrink_amt` and `inter_stocktake_sales_amt` fields will be reset by this program and re-start again to accumulate towards the next stock count.

Note: `inter_stocktake_shrink_amt` and `inter_stocktake_sales_amt` fields on `month_data` are used by `stkdyly.pc` to calculate the book stock value and the actual shrinkage amount for a stock count.

After all threads for this program have finished processing, the `prepost` module must be launched by the scheduler to update the various end-of-month columns on `SYSTEM_VARIABLES`.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DAILY_DATA	Yes	Yes	No	No	No
DEPS	Yes	Yes	No	No	No
HALF_DATA_BUDGET	Yes	Yes	No	No	No
MONTH_DATA	Yes	Yes	Yes	Yes	No
PERIOD	No	Yes	No	No	No
STAKE_HEAD	Yes	Yes	No	No	No
STAKE_PROD_LOC	Yes	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	Yes	No
V_RESTART	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE 3 (monthly)

Scheduling Diagram: Can run any time after end-of-month date
Must run salweek first before running salmth

Pre-Processing: N/A

Post-Processing: salmth_post()

Updates system variables to set the stock ledger calendar ahead to the next month for processing. All stock ledger calendar dates are moved forward to indicate that the current month's stock ledger processing has completed.

Threading Scheme: STORE_WH
V_restart_store_wh

Restart Recovery

```
SELECT month_data.dept,  
  
       month_data.class,  
  
       month_data.subclass,  
  
       month_data.store,  
  
       month_data.wh,  
  
       month_data.currency_ind,  
  
       NVL(month_data.opn_stk_cost,0),  
  
       NVL(month_data.opn_stk_retail,0),  
  
       NVL(month_data.inter_stocktake_sales_amt,0),  
  
       NVL(month_data.inter_stocktake_shrink_amt,0),  
  
       NVL(month_data.stocktake_mtd_sales_amt,0),  
  
       NVL(month_data.stocktake_mtd_shrink_amt,0),  
  
       NVL(month_data.htd_gafs_cost,0),  
  
       NVL(month_data.htd_gafs_retail,0),  
  
       NVL(month_data.stocktake_bookstk_cost, 0),  
  
       NVL(month_data.stocktake_bookstk_retail, 0),
```

```

        NVL(month_data.stocktake_actstk_cost, 0),
        NVL(month_data.stocktake_actstk_retail, 0),
        ';' || TO_CHAR(month_data.dept) ||
        ';' || TO_CHAR(month_data.class) ||
        ';' || TO_CHAR(month_data.subclass) ||
        ';' || TO_CHAR(month_data.store)
FROM month_data,
     v_restart_store_wh rv
WHERE month_data.half_no = :half_no
      AND month_data.month_no = :month_in_half
      AND rv.driver_value = month_data.store
      AND rv.driver_name = :ora_restart_driver_name
      AND rv.num_threads = :ora_restart_num_threads
      AND rv.thread_val = :ora_restart_thread_val
      AND (month_data.dept > NVL(:ora_restart_dept, month_data.dept - 1) OR
           (month_data.dept = :ora_restart_dept AND
            (month_data.class > :ora_restart_class OR
             (month_data.class = :ora_restart_class AND
              (month_data.subclass > :ora_restart_subclass OR
               (month_data.subclass = :ora_restart_subclass AND
                (month_data.store > :ora_restart_store)))))))
ORDER BY month_data.dept,
         month_data.class,
         month_data.subclass,
         month_data.store,
         month_data.currency_ind;

```

Program Flow

N/A

Shared Modules

STKLEDGR_ACCTING_SQL.RETAIL_METHOD_CALC: performs stock ledger stock and gross margin calculations using the retail accounting method.

STKLEDGR_ACCTING_SQL.COST_METHOD_CALC: performs stock ledger stock and gross margin calculations using the cost accounting method.

CAL_TO_454_LDOM: determines the 454 last-day-of-month from current calendar date.

CAL_TO_CAL_HALF: determines the half number based on current date.

CAL_TO_CAL_LDOM: determines last-day-of-month based on current date.

CAL_TO_454_WEEKNO: determines week number in 454 half from current date.

CAL_TO_CAL_WEEKNO: determines calendar week number from current date.

CAL_TO_454: determines 454 calendar date from current date.

HALF_TO_CAL_FDOH: determines the first-day-of-half from the current half number.

HALF_TO_CAL_LDOH: determines the last-day-of-half from the current half number.

HALF_TO_454_FDOH: determines the 454 first-day-of-half from current half number.

HALF_TO_454_LDOH: determines the 454 last-day-of-half from current half number.

Function Level Description

First check if there are unprocessed “Unit & Dollar” type of stock count, if there are any, stop processing and give user an error message

Main process :

Loop through all subclass/location on month_data for current month (month to be processed). For each month_data record fetched :

read profit_calc_type and purchase_type from deps table

read shrinkage_pct from half_data_budget table

check if Unit & Dollar type of stock count occurs during the month

sums the DAILY_DATA records by transaction type for all records with date in the current month

if purchase_type = 1 (consignment department)

gross_margin_amt = purch_retail - purch_cost

else

if cost method (profit_calc_type = 1)

call the package function stkledgr_accting_sql.cost_method_calc to calculate week's closing stock at cost and gross margin

else

call the package function stkledgr_accting_sql.cost_method_calc to calculate week's closing stock at retail and cost and gross margin

call function update_month_data

Update month_data for current month

Insert a row for next month on month_data

copy current month's closing stock to be opening stock for next month

copy inter_stocktake_shrink_amt and inter_stocktake_sales_amt from current month to next month

if current month_no = 6 (last month of the half) reset GAFS cost and retail :

 htd_gafs_cost and retail of next month = cls_stk_cost and retail
of current month, respectively

 else

 copy htd_gafs_cost and retail from current month to next month

I/O Specification

N/A

Technical Issues

N/A

Purge Stock Ledger Transactions [salprg]

Design Overview

This program deletes tran_data records that are older than the user defined number of retention days, which is stored in system_options.tran_data_retained_days_no.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
TRAN_DATA	No	No	No	No	Yes

Scheduling Constraints

Processing Cycle: PHASE AD-HOC (daily)

Scheduling Diagram: Advantageous to run this module prior to phase 3 since this will improve select operations from

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

N/A

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Stock Ledger Stage [salstage]

Design Overview

In order to make the rollup and extraction of the stock ledger transaction data the most flexible, this program will move the data on the tran_data to the if_tran_data staging table. This will enable the processes that are writing records to tran_data to continue in a seamless manner, whereas the processes that roll the data up to a different level or extract the data to external systems can work without affecting batch timetables.

This process will be achieved by locking the tran_data table and moving all of the data to the staging table. The original tran_data table will be emptied and the lock on the table will be released. Before this processing occurs, the staging table will first be emptied to ensure that data is not processed twice.

The data on the tran_data and if_tran_data tables is very transitional. These tables will fill up and be truncated at least once a day if not several times per day.

Scheduling Constraints

Processing Cycle: anytime (this will take a snapshot of data on tran_data)

Scheduling Diagram: N/A

Pre-Processing: inventory processing

Post-Processing: saldly, extractions against if_tran_data, salapnd

Threading Scheme: Threading will be implicit via the use of the Oracle Parallel Query Option. The insert/select query should be tuned for each specific environment to achieve the best throughput.

Restart Recovery

No specific restart/recovery scheme needs to be defined because of the limited scope of the module. However, in cases where the tran_data table is very large, a particularly large rollback segment may be specified to reduce the risk of exceeding rollback segment space. This will depend on the size of normal rollback segments and the size of the tran_data table.

Program Flow

N/A

Shared Modules

N/A

Function Level Description

Init():

The rollback segment should be specified to a large enough size so that the entire rollup will complete in a single transaction.

Process()

This function first call the drop_index function to remove the indexes on the if_tran_data table. It then calls the truncate_table function to remove any data in if_tran_data so it is not processed twice. Then it calls the lock_table function to ensure that no changes take place while the move is taking place. Once the lock is placed on the tran_data table, the build_stage function is called. After the data is moved to the if_tran_data table the truncate_table function is called to remove the old data from the tran_data table. Next the indexes are rebuilt in the if_tran_data table. Finally the lock is removed from the tran_data table.

Drop_index()

This function drops the indexes in the if_tran_data table.

Lock_table()

This function places or removes a lock on a table passed in. It will place or release the lock depending on an input parameter.

Build_stage()

This function builds the if_tran_data by moving all of the data from the tran_data table onto the if_tran_data table. This is done using an insert/select statement with a hint for parallelism. The degree of parallelism should be customized to each run-time environment. After the data is moved the degree of parallelism is set to zero.

Rebuild_indexes()

This function rebuilds the indexes on the if_tran_data table. The indexes are built in parallel, unreciverable. After the indexes are built their degree of parallelism is set to zero.

Truncate_table()

This function truncates the table that is passed in as an input parameter using dynamic SQL.

I/O Specification

N/A

Technical Issues

N/A

Weekly Stock Ledger Processing [salweek]

Design Overview

This program processes all weeks that are in the month for which month-end process has not been run, up to the current week. It rolls up data on DAILY_DATA to the corresponding subclass/location/week record on WEEK_DATA. The closing stock value and gross margin are calculated by calling the appropriate package function, based on the accounting method (i.e. profit calculation type) chosen for the department. Closing stock value for a processed week becomes opening stock value for the next week. A WEEK_DATA row for next week is inserted if it does not already exist.

If a stock count occurs during a certain week, stkdly.pc would have already updated that week's week_data.stocktake_adj_cost and retail fields with the difference between book stock value and actual stock value (results of stock count). This stocktake_adj_cost and retail are used to adjust weekly closing stock value in this program. The closing stock value for all the weeks that follow will also be adjusted, due to the fact that this week's closing stock will be rolled into next week's opening stock.

In addition, this program calculates a shrinkage amount as follows :

If budget shrinkage indicator = 'Y' :

shrinkage amount = budgeted shrinkage percent * sales amount

for each week processed

else

shrinkage amount = - (stock_adj_cost or retail)

depending on cost or retail method is used

The above calculated shrinkage amount will then be used to reduce the closing stock for each week processed.

This program not only runs every week-end, it must also run immediately before salmth runs, since salmth could be run any day during the week. This is to update week_data with tran_data record that comes in between the last time salweek runs and when salmth runs. The only difference between these two salweek runs is that : a WEEK_DATA row for next week is inserted only if this program runs on week-end.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DAILY_DATA	Yes	Yes	No	No	No
DEPS	No	Yes	No	No	No
HALF_DATA_BUDGET	No	Yes	No	No	No
PERIOD	No	Yes	No	No	No
CALENDAR	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	Yes	No
WEEK_DATA	No	Yes	Yes	Yes	No

Scheduling Constraints

Processing Cycle: Stock Ledger

Scheduling Diagram: Can be run anytime after the week has completed.

Pre-Processing: Saldly

Post-Processing: Salmth (if applicable)

Threading Scheme: Department via v_restart_dept view

Restart Recovery

Restart is on dept, class, and subclass. Clients with substantial number of stores may consider including store and warehouse in the restart logic.

Program Flow

N/A

Shared Modules

STKLEDGR_ACCTING_SQL.RETAIL_METHOD_CALC: performs stock ledger stock and gross margin calculations using the retail accounting method.

STKLEDGR_ACCTING_SQL.COST_METHOD_CALC: performs stock ledger stock and gross margin calculations using the cost accounting method.

454 Calendar functions

Function Level Description

Main

Init

- Initialize restart variables.
- Collect system date information
- Get_weeks_to_process function callout to fill array with week information since last eom
- Get_half_info function callout to determine if data will include more than 1 454 half
- Size_location function callout to determine the size for the location array (#loc * # halves) to hold budgeted shrinkage
- Load_dept function callout to size and fill department array with profit calc and purchase type info
- Size_arrays function callout to size all other arrays according to value from restart API
- Setup_daily_zeros_array function callout to create a single dimensional array filled with zeros for weeks when no daily transactions exist on daily_data

Get_half_info

- if the start of the current half (the half belonging to the system date) is later than the last end of month date, then more than one half has been spanned.
- Determine value of last half to return to calling function.

Get_weeks_to_process

- Size_weeks function callout to size the week array appropriately
- select date information from the 1st week to process (1st week after last end of month) to the current end of week date
- cal_to_454_half function callout to gather 454 calendar information about each date.
- Populate each row of the week array with each week's information

Cal_to_454_half

- CAL_TO_454_HALF PL/SQL procedure callout for centralized function

Size_week

- use the number of weeks value from the init function to appropriately size the array that will hold weekly calendar information from the last end of month date to the current end of week date.

Setup_daily_zeros_array

- apply values of zero to a one-dimensional array that will be used to recalculate weekly records when no daily records exist.

Process

- The data from week_data and daily_data should be scrolled through to evaluate whether daily stock ledger records exist for each week record. Week records should always exist for each department, class, subclass, store, warehouse, currency indicator and end of week category, however daily records may not exist. If daily records exist or if stock take adjustments have been recorded (on weekly records) then the weekly stock ledger data should be reevaluated and updated for that week and all subsequent weeks.
- open_cursors function callout to open daily and weekly cursors.
 - Weekly data will be selected from the week_data table to drive the program's processing. All weekly data will be selected that has an end of week date that is after the last closed end of month date. Data will be grouped and ordered by department, class, subclass, store, warehouse, currency indicator, and end of week date.
 - Daily data will be selected from the daily_data table to match the data selected from the week_data cursor. All daily_data will be selected that has an end of week date that is after the last closed end of month date. Data will be grouped and ordered by department, class, subclass, store, warehouse, currency indicator, and end of week date.
- Fetch_week_cursor function callout to handle the fetching of week data into arrays for processing.
- Fetch_day_cursor function callout to handle the fetching of day data into arrays for processing.
- First_week_handler function callout to evaluate and process the first week_data record for a stock ledger category (dept, class, subclass, store, warehouse, currency indicator, and end of week date).
- Day_handler function callout will evaluate fetched records and process them.

- Last_week_handler function callout will process the last records in the stock ledger category and insert a new record for the next week.
- Post_arrays function callout will post the filled and cached insert and update arrays to the database.

Open_cursors

- Open the week_data cursor.
- Open the daily_data cursor.

Fetch_week_cursor

- fetch week_data cursor into arrays.
- Because daily_data and week_data records may not match up exactly (daily_data records may not exist for each week), the daily and week data should be fetched separately.

Fetch_day_cursor

- fetch daily_data cursor into arrays.
- Because daily_data and week_data records may not match up exactly (daily_data records may not exist for each week), the daily and week data should be fetched separately.

First_week_handler

- cache beginning of week data, including opening stock position. The stock position for the first week of the stock ledger category will be correct because of the closing of the previous month.
- Get_dept_info function callout to fill department array with profit calculation type and purchase type information.
- Set_start_values function callout to populate facts for first week in stock ledger category
- Cache stock ledger category information (dept, class, subclass, store, wh and currency indicator) for use in processing.
- Evaluate whether items should be posted to the database from the arrays and the data committed. The criteria will be based on whether the restart commit counter (of the restart/recovery API) has been reached or exceeded. Further, because this is the first week of a stock ledger category, all of the information in the arrays will be of complete stock ledger categories.
- Post_arrays function callout will post the cached array data to the database.

- Restart_commit function callout will commit the data that has been thus far posted to the database.

Set_start_values

- initialize first values of information that will be re-calculate and accumulated via packages for the stock ledger category.
 - Cls_stk_cost
 - Htd_gafs_cost
 - Retail shrinkage
 - Cost shrinkage
 - Cumulative markon percent
 - Gross margin amount

day_handler

- This function will be called after the first week of the stock ledger category week_data information has been evaluated.
- Spool through daily and week arrays until all weeks of the stock ledger category have been processed.
- If no daily records exist for the stock ledger category exist, then evaluate the rest of the weekly records to determine if stock take adjustments have been applied to any of the weeks. If adjustments exist for any of the weeks, then the stock ledger information will have to be recalculated and updated (week_data) for that week and all subsequent weeks in the category. Values of zero will be passed for what would have been fetched from the daily_data information.
- If any daily_data records exist for any weeks, then that week and all subsequent weeks will be processed. The cached and cumulative data will be rolled up across the weeks, the daily_data information (zeros for weeks without any daily records), and the stock take adjustments will be used to recalculate the stockledger.
- Set_up_daily_values function callout will populate the variables that will be used in the package call for stock ledger recalculations. Either the current fetched information or zeros will be passed as parameters depending on whether daily data exists or not for the current week.
- increment_week_data_week function callout
- fetch_week_cursor function callout to fetch new batch of data from the week_data table

- `make_fnc_calls` function callout to call the stock ledger calculation packages that recalculate the stock information for the week based on the daily info, stock take adjustments, and the opening stock info.
- `Fill_update_array` function callout will populate the `week_data` update array with the new values for the current week
- If the weeks being evaluated span 454 halves, then reset the half to date stock variables (gafs cost & retail) for the new half.

Setup_daily_values

- either `daily_data` values or zeros will be passed in as parameters to this function.
- The passed data will be copied into holder variables to be used in stock ledger function calls and for rolling accumulations for the stock ledger category.

Make_fnc_calls

- `get_location_info` function callout will get the appropriate shrinkage rate for the current department, location, and half.
- For non-consignment items, the `daily_data` values (or zeros) will be passed with the shrinkage rate to the stock ledger calculation package
- `Cost_package_call` function callout will be called if cost method of accounting is used.
- `Retail_package_call` function callout will be called if retail method of accounting is used.
- For consignment items, the stock ledger packages will not be called, but the gross margin amount will be set for the update (purchase retail – purchase cost).

Fill_update_array

- `daily_data` info, calculated stock ledger positions, and open/closing stock position information is copied into the `week_data` update array
- for non-consignment items, only the following items will be updated
 - open stock retail
 - open stock cost
 - purchase costs
 - purchase retail
 - net sales cost

- net sales retail (includes vat)
 - net sales retail excluding vat
 - sales units
 - gross margin amount
- for consignment items all data will be updated.
 - For profit calculations of type direct cost, half to date retail, gafs cost and cumulative markon percent will be updated as null
 - Post_update function callout will post the array to the database if it is full.

Retail_package_call

- STKLEDGR_ACCTING_SQL.RETAIL_METHOD_CAL function callout will (via a PL/SQL package) use the retail method of accounting to calculate the stock ledger values for the current week's data.
- The accumulated values will be updated for the next week in the stock ledger category
 - Cls stk retail
 - Cls stk cost
 - Half to date gafs retail
 - Half to date gafs cost
 - Gross margin amount
 - Cumulative markon percent
 - Shrinkage retail

Cost_package_call

- STKLEDGR_ACCTING_SQL.COST_METHOD_CAL function callout will (via a PL/SQL package) use the cost method of accounting to calculate the stock ledger values for the current week's data.
- The accumulated values will be updated for the next week in the stock ledger category
 - Cls stk retail
 - Cls stk cost
 - Gross margin amount
 - Shrinkage cost

Last_week_handler

- last_ins_update function callout will populate either insert or update arrays for the last week in the process.
- Progress the week_data to the first week of the next stock ledger category.
- Fetch_week_cursor function callout will fetch new batch of week_data data if all of the elements of the current fetch have been processed.

Last_ins_update

- if the last week of the week_data information from the current stock ledger category is equal to the maximum process date, then populate the update array with the current week's data, otherwise populate the week_data insert array with the current week's data.
- Fill_last_week_upd function callout will populate the week_data update array
- Populate_insert function callout will populate the week_data insert array

Populate_insert

- get the 454 information for the last week
- CAL_TO_454 function callout will calculate the 454 calendar information for the passed date
- For all item types, copy the following fetched and calculated information into the insert array:
 - Dept, class, subclass, store, warehouse, currency indicator, end of week date
 - Half number, month number
- For non-consignment items with a normal purchase type copy the following additional information:
 - Half to date gafs retail as calculated
 - Half to date gafs cost as calculated
- For non-consignment items with a direct cost purchase type, copy the following additional information:
 - Half to date gafs retail as null
 - Half to date gafs cost as calculated

- For consignment items, copy the following additional information:
 - Opening stock retail as zero
 - Opening stock cost as null
 - Half to date gafs retail as null
 - Half to date gafs cost as null
- Post_insert function callout will post the arrays to the database if they are full

Fill_last_week_upd

- for normal purchase types
- for retail profit calculation type
 - half to date gafs retail will equal the calculated value
 - half to date gafs cost will equal the calculated value
- for cost profit calculation type
 - half to date gafs retail will be null
 - half to date gafs cost will be null
- for consignment purchases
 - half to date gafs retail and cost, open stock retail, and open stock cost will all be null
 - post_last function callout to post update array to the database if the array is full

post_arrays

- post_insert function callout to post array insert to week_data to database
- post_update function to post array update of week_data to database
- post_last function to post array update of the last week of week_data info to the database

post_insert

- embedded SQL call to perform bulk insert of array to week_data

post_update

- embedded SQL call to perform bulk update of array to week_data

post_last

- embedded SQL call to perform bulk update of array to week_data
- size_arrays
- allocate system memory for arrays in use
 - the number of rows of each array should be determined by the counter (commit_max_ctr) on the restart_control table. The value will be obtained in the init via the function call to restart_init
 - the zero array will be sized to have only one row.

Size_location

- this function counts all of the stores and warehouses in the system and sizes the array to hold shrinkage info at the location/half/department level.
- If more than one half will be spanned by processing then, then the size will be twice the number of locations, otherwise it will be equal to the number of locations.
- If more than two halves are spanned then a fatal error will occur. This means that the salweek program hasn't been run for more than a year.
- Allocate memory for the array elements using the sizing method described above.

Load_dept

- this function loads the profit_calc_type and purchase type for every dept in the system into an array that will be used by the main processing as it moves through the week_data.
- The size of the array will be determined by the total number of departments in the system.
- The number of rows in the array is limited to 32K. if more than 32K departments exist, the code will have to be modified to handle this.
- Size_deps function callout will physically allocate the memory necessary to accommodate the array.
- The array is filled with department, profit calculation type, and purchase type ordered by department.

Size_deps

- this function physically allocates memory for the department array based on the number of departments in the system.

Get_dept_info

- this function will be called to get the department level info for the stock ledger category.
- The array will be searched for the appropriate department and the profit calculation and purchase type will be obtained.

Load_location

- This function loads the shrinkage rate for each location in a dept for a given half.
- When the department changes, the location/half arrays will be filled with all of the shrinkage and location info.
- An array will be used for each location type (store and warehouse).

Get_location_info

- this function steps through the location array(s) to get the shrinkage info for the location/half for each department.
- Load_location function callout will be called to fill the location array(s) if the department has changed.

I/O Specification

N/A

Technical Issues

Due to the transaction volumes that this program must process, array processing has been heavily utilized in making Oracle system calls.

The following describe the arrays created for Oracle calls

- Daily_data, select array
- Week_data, select, update and insert arrays

Additional structures of arrays:

- Week and 454 calendar information (for weeks from last end of month to today)
- department level array for searching for profit calc type and purchase type information
- location/half cache for searching for budgeted shrinkage information

[samastersfm.pc]

Design Overview

This module will transmit additions, modifications and/or deletions of Retek location and item reference data to the Oracle Site Fuel Management (SFM) application.

Table	Operations Performed			
	Select	Insert	Update	Delete
sa_if_location	Yes	No	No	Yes
sa_if_item	Yes	No	No	Yes
store	Yes	No	No	No
win_skus	Yes	No	No	No
sa_system_options	Yes	No	No	No
sa_store_data	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Anytime – Sales Audit 9.0 is a 24/7 system.

Scheduling Diagram: This module can be executed on demand, but needs to be scheduled before saexpfm.

Threading Scheme: N/A

Restart Recovery

Restart/Recovery is not used in this module.

Driving cursors:

To select store records:

```
EXEC SQL DECLARE c_loc_data CURSOR FOR

SELECT s.store, s.store_name, s.currency_code

FROM sa_if_location loc, sa_store_data sd, store s

WHERE loc.location = sd.store

AND loc.location = s.store

AND loc.action_type = :ls_action_type
```

```
AND sd.system_code = :ls_system_code
```

```
AND sd.imp_exp = 'E'
```

```
ORDER BY s.store;
```

To select item records:

```
EXEC SQL DECLARE c_item_data CURSOR FOR
```

```
SELECT sku.sku, sku.sku_desc, si.action_type
```

```
FROM sa_if_item si, win_skus sku, sa_system_options so
```

```
WHERE si.item = sku.sku
```

```
AND sku.dept = so.fuel_dept
```

```
ORDER BY sku.sku;
```

The sa_if_location and sa_if_item tables are populated by triggers when records are written to the sa_store_data and win_skus tables.

Shared Modules

SFM API:

- eas_retail_sites_pub.update_retail_site is used to update store data on the SFM tables.
- eas_articles_pub.create_article is used to add an item record to the SFM tables.
- eas_articles_pub.update_article is used to update an item record on the SFM tables.
- eas_articles_pub.delete_article is used to delete an item record from the SFM tables.

Function Level Description

Init ()

Perform any variable initialization necessary for processing.

Process ()

For each record returned from the sa_if_location table, determine whether or not it is a location that sells fuel and therefore should be exported to SFM. This is accomplished by selecting from the sa_store_data where the system code is "SFM" and the import export indicator is "E"xport.

If it is a store with an export of fuel data, call the appropriate SFM package for the given action code on the sa_if_location entry.

Call the update_retail_site package if the action code is "M"odify

After processing all store records delete all records in sa_if_location table.

For each record returned from the sa_if_item table, determine whether or not the given item is in the fuel department. This is accomplished by selecting the sku's department from the RMS win_skus table, and comparing it to the fuel department defined on the ReSA sa_system_options table.

If it is a fuel item, call the appropriate SFM package for the given action code on the sa_if_item entry.

Call the create_article package if the action code is "A"dd

Call the update_article package if the action code is "M"odify

Call the delete_article package if the action code is "D"elete

After processing all item records delete all records in sa_if_item table.

Final ()

Clean up any memory used.

I/O Specification

The SFM package specifications can be found in the Oracle Back Office APIs document (SFMAPID1A.doc).

[sapreexp.pc]

Design Overview

When a user modifies or revises a transaction through the Sales Audit user application, numerous totals will be affected through re-totaling. The sales audit pre-export module is designed to compare the latest prioritized version of each total defined for export with the version that was previously sent to each system. If they are the same, an SA_EXPORTED entry should be created for the total for that particular system so that the same value will not be exported twice. By determining which totals have not changed since the last export date time (SA_EXPORTED_REV), this module will then create entries on SA_EXPORTED to prohibit any third party application from receiving multiple export revisions.

Table	Operations Performed			
	Select	Insert	Update	Delete
Sa_store_day	Yes	No	No	No
Sa_export_log	Yes	No	No	No
Sa_exported	Yes	Yes	No	No
V_sa_total_value	Yes	No	No	No
Sa_exported_rev	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Anytime – Sales Audit 9.0 is a 24/7 system.

Scheduling Diagram: This module should be run after the ReSA auditing process and before any export processes.

Threading Scheme: N/A

Restart Recovery

The logical unit of work for this module is defined as a unique store/day combination. Records will be fetched, updated and inserted in batches of g_l_restart_max_counter. Only two commits will be done. One to establish the store/day lock (this will be done by the package) and one at the end after a store/day or store/day/total has been completely processed.

Driving cursor:

```
SELECT DISTINCT a.store_day_seq_no
FROM   sa_export_log a
WHERE  a.status = 'R'
AND    a.seq_no =
      (SELECT MAX(b.seq_no)
       FROM   sa_export_log b
       WHERE  b.store_day_seq_no = a.store_day_seq_no)
ORDER BY a.store_day_seq_no;
```

Work cursors:

There are two work cursors. The first work cursor will get totals that were previously exported. It is used in the `fetch_already_exported()` function.

```
EXEC SQL DECLARE c_already_exported CURSOR FOR
```

```
/* select all totals that have been exported already */
```

```
SELECT v.total_seq_no,
       v.value * :pl_multiplier,
       el.system_code
FROM   v_sa_get_total v,
       sa_total_usage tu,
       sa_export_log el,
       sa_exported_rev er
WHERE  v.store_day_seq_no = :is_store_day_seq_no
AND    el.store_day_seq_no = v.store_day_seq_no
AND    el.status = :SAES_R
AND    el.seq_no > 1
AND    tu.total_id = v.total_id
AND    tu.total_rev_no = v.total_rev_no
```

```
AND substr(tu.usage_type, 1, 3) = el.system_code

AND er.total_seq_no = v.total_seq_no

AND substr(er.system_code, 1, 3) = el.system_code

AND er.rev_no = v.value_rev_no

AND er.exp_datetime =

/* select the last one exported */

(SELECT MAX(exp_datetime)

FROM sa_exported_rev

WHERE total_seq_no = v.total_seq_no

AND substr(system_code, 1, 3) = el.system_code)

ORDER BY 1, 2, 3;
```

The second work cursor retrieves all the totals that are ready for export.

```
EXEC SQL DECLARE c_ready_to_export CURSOR FOR
```

```
/* select all totals ready for export */

SELECT v.total_seq_no,

       v.value * :pl_multiplier,

       el.system_code

FROM v_sa_get_total v,

     sa_total_usage tu,

     sa_export_log el

WHERE v.store_day_seq_no = :is_store_day_seq_no

AND v.precedence =

     (SELECT min(v2.precedence)

      FROM v_sa_get_total v2

      WHERE v2.total_seq_no = v.total_seq_no)

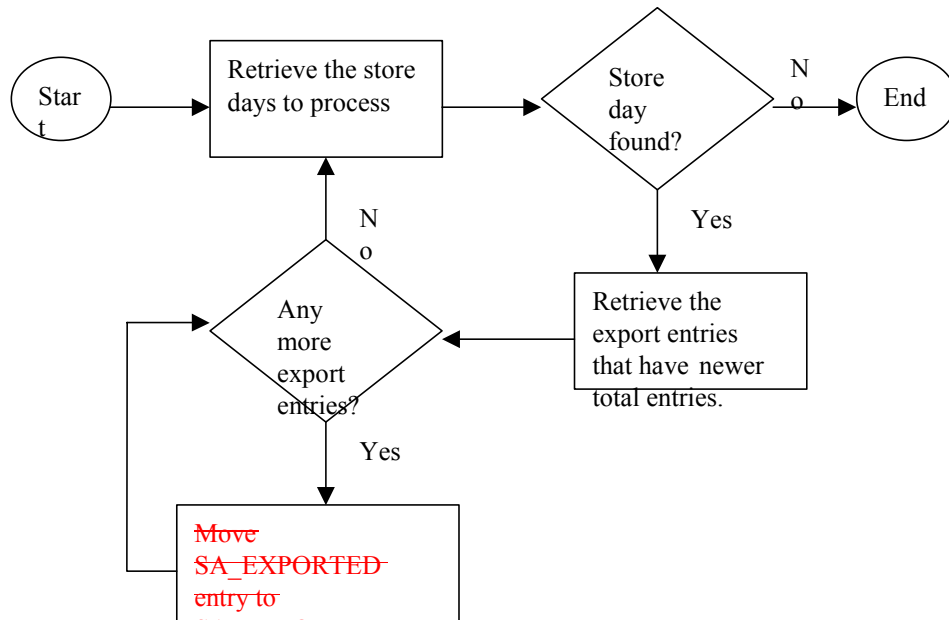
AND v.value_rev_no =
```



```
(SELECT max(v2.value_rev_no)
FROM v_sa_get_total v2
WHERE v2.total_seq_no = v.total_seq_no
AND v2.precedence = v.precedence)
AND el.store_day_seq_no = v.store_day_seq_no
AND el.status = :SAES_R
AND el.seq_no > 1
AND tu.total_id = v.total_id
AND tu.total_rev_no = v.total_rev_no
AND substr(tu.usage_type, 1, 3) = el.system_code
AND NOT EXISTS
(SELECT 1
FROM sa_exported
WHERE total_seq_no = v.total_seq_no
AND substr(system_code, 1, 3) = el.system_code);
```

The program then compares the values retrieved by these two cursors and if a match is found, then an entry in the sa_exported table is created to prevent this total value from being re-exported.

Program Flow



Shared Modules

ReSA Batch Library functions used:

- `get_lock` – used to establish a read lock on a store/day.
- `release_lock` – used to release a store/day lock.

Function Level Description

`init()`

1. Call `init_exported_array`.
2. Call `retek_init`

`process()`

1. The Driving Cursor should retrieve the first store day to process.
2. Attempt to lock the store/day with a call to `get_lock` library function. If this fails, go on to the next store/day.
3. Call the procedure `process_store_day_exported`.
4. Release lock and commit store day work completed.

5. Loop through driving cursor.

final()

1. Call free_exported_array.
2. Call retek_close for a final commit.
3. Call retek_refresh_thread to refresh the current thread.

Process_store_day_exported()

1. Call fetch_already_exported() to retrieve the first work cursor.
2. Retrieve the working cursor for the current store day.
3. Find if the current row from the second work cursor exists in the data fetched by the first work cursor by calling already_exported_lookup().

If it does exist, insert into the sa_exported table using the values from the sa_exported_rev table for the total_seq_no and system_code. Else do nothing.

fetch_already_exported()

This function uses a global variable to hold all of the data fetched from the first work cursor. The array holding this data is resized as necessary.

already_exported_lookup()

This function uses the C library function bsearch to search the array populated by fetch_already_exported() for data that matches the parameter passed in. The actual comparison is performed in nodecmp().

nodecmp()

This function concatenates the two pa_export_data rows passed in as parameter and compares them using strncmp.

init_exported_array()

Allocate memory for il_size items for the working cursor arrays.

Free_exported_array()

Free the working cursor arrays from memory.

I/O Specification

There are no input or output files for this module.

[sapurge.pc]

Design Overview

This program will be run daily to control the size of the tables in the sales audit database. Older information will be deleted to ensure optimal performance of the system as a whole.

Different kinds of data need to be kept in the system for different amounts of time. Transactions, and all associated transaction details, and Totals calculated or reported for a store day will be deleted when they meet the following criteria:

- The Business Date for those transactions and totals is older than or equal to today's date minus the days_before_purge parameter setup on the sa_system_options table.
- No locks exist on the store/day.
- One of the two following statements is true for the store/day:
 1. Fully loaded, and all errors either corrected or overridden (sa_store_day.audit_status is 'A' (Audited) and sa_store_day.data_status = 'F' (Fully loaded)). In addition, there are no outstanding exports (records for the store/day in the sa_export_log table where sa_export_log.status = 'R' (Ready for export)).
 2. Never loaded (sa_store_day.audit_status is 'U' (Unaudited) and sa_store_day.data_status = 'R' (Ready for import)).

Flash Sales data will be deleted when it meets the following criteria:

- Date is two years before today's date minus the days_before_purge parameter setup on the sa_system_options.
- Company open and close dates will also need to be kept for two years plus days_before_purge so that the historical comparisons in flash sales reporting carry the appropriate weight.

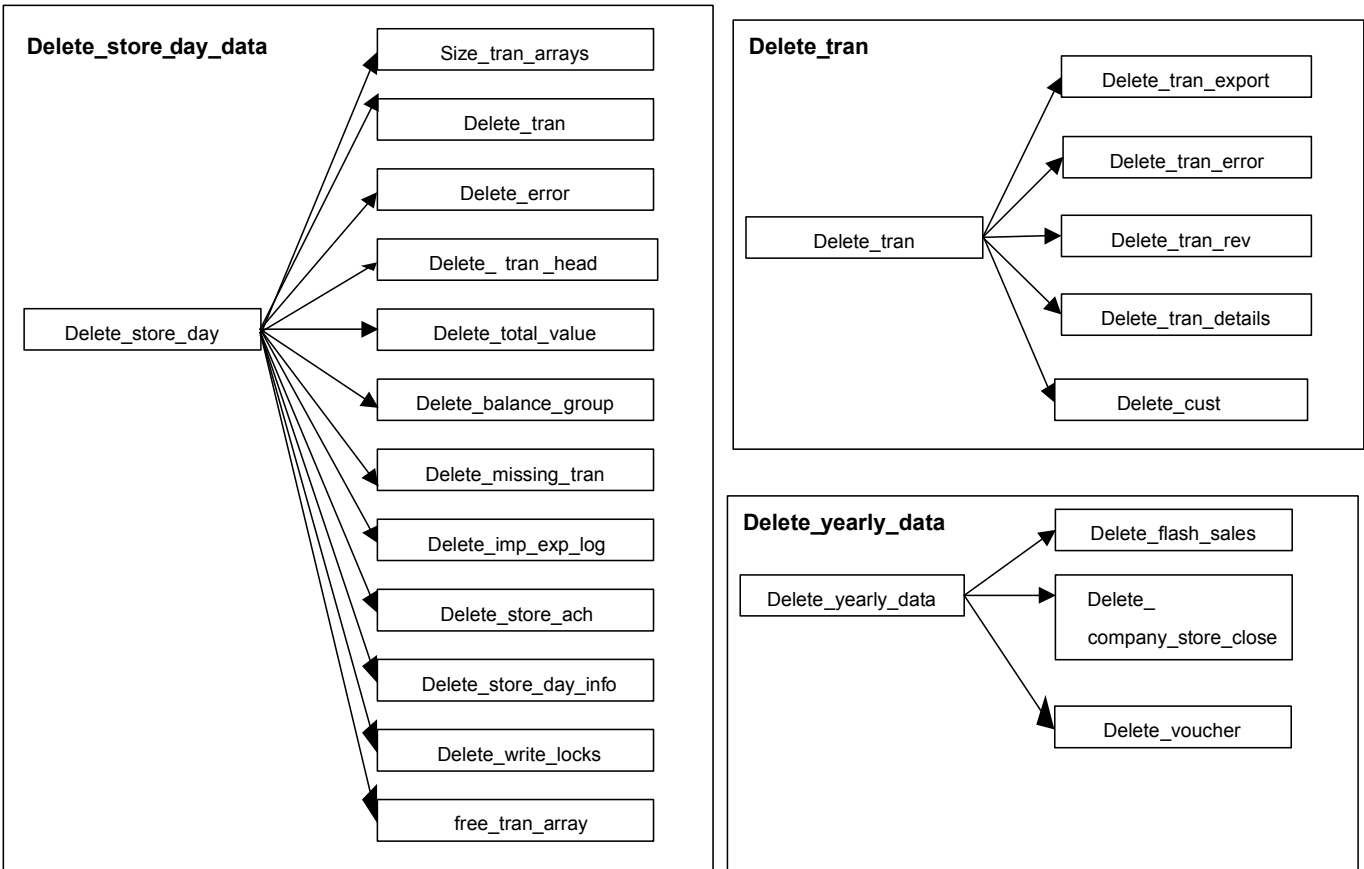
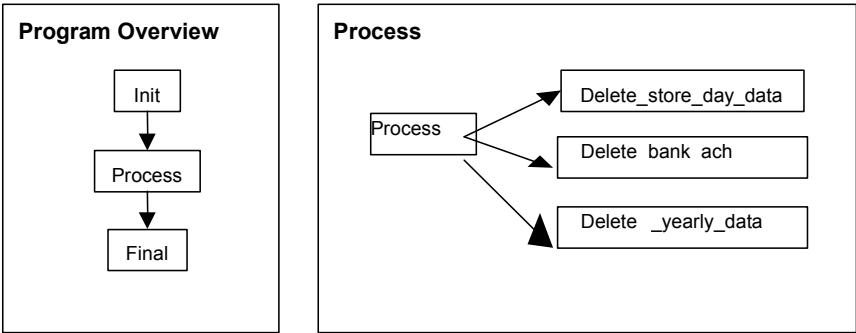
Voucher data will be deleted when it meets the following criteria:

The redeemed date or the escheat date for the specific voucher type is before today's date minus the purge_no_days on the sa_voucher_options table for the corresponding voucher type.

The program can also take in a list of store_day_seq_no to delete. For example, the command line could be: sapurge userid/passwd 1000 1001 1002, where 1000, 1001 and 1003 are store_day_seq_nos that the user wants to delete. These must also meet the criteria defined above. If a store_day_seq_no is passed to this program, but does not meet the criteria, an error will be written out to the error log.

An output file will be created to store a record for each store and business date that was purged. The file name must be passed in at the command line as a parameter to sapurge.

sapurge.pc



Stored Procedures / Shared Modules (Maintainability)

Libretek.a library functions:

- Retek_init/rettek_close – to initialize and close Retek's restart/recovery module
- FetchVdate – to get the current date and time
- FetchSaSystemOptions – to get days before purge value from ReSA system options table.

Input Specifications

Table-To-Table

The list of tables from which data is going to be deleted are:

- sa_store_day
- sa_tran_head
- sa_tran_tax
- sa_tran_item
- sa_tran_disc
- sa_tran_tender
- sa_exported
- sa_tran_head_rev
- sa_tran_tax_rev
- sa_tran_item_rev
- sa_tran_disc_rev
- sa_tran_tender_rev
- sa_exported_rev
- sa_customer
- sa_cust_attrib
- sa_balance_group

- sa_total
- sa_pos_value
- sa_sys_value
- sa_store_value
- sa_hq_value
- sa_total_value_rev
- sa_error
- sa_error_rev
- sa_store_error
- sa_missing_tran
- sa_import_log
- sa_export_log
- sa_store_ach
- sa_bank_ach
- sa_flash_sales
- sa_company_close
- sa_company_excep
- sa_store_close
- sa_voucher

Function Level Description

Init – This function will fetch the `system_options.days_before_purge` and the `period.vdate` into global variables to be used throughout the program. This function will also call `restart_init` to begin the restart/recovery bookmark process. The file to store purged store/days will need to be opened.

If a list of `store_days` are passed at the command line, then it will copy them into an array of the appropriate size.

CopySDArray – This function is called from Init and copies the list of `store_days` passed at the command line into an array.

Process – This function will call the function `process_delete_store_day_data`, and if no `store_day_seq_nos` are passed at the command line, it will also call the function `delete_yearly_data`, each of which will deal with records from different 'time' periods. `process_delete_store_day_data` deletes all associated detail records that correspond to a single store/day. `delete_yearly_data` deletes records for which two years of history is needed.

`process_delete_store_day_data` – This function will call functions to delete information that grows by store/day.

If a list of `store_day_seq_nos` are passed at the command line, then only these `store_days` will be passed in to the function to determine which `store_days` to delete, otherwise all will be considered for deletion.

The driving cursor in this function will select the outdated `sa_store_day.store_day_seq_no`. A `sa_store_day.store_day_seq_no` is outdated when it's `sa_store_day.business_date` is equal to the `period.vdate` – `sa_system_options.days_before_purge`.

Also, the program will only delete those store/days without errors.

Before any purging occurs, the function should set `sa_store_day.data_status` to 'G' for Purging. This lets the user know that he/she should not be locking the data.

This function must also try to obtain a write lock by using the `get_lock()` function in the sales audit library. If this function returns a 1, then the store/day is locked. The store/days that meet this criteria should be fetched into an array. If a 0 is returned by `get_lock()`, then a lock on this store/day is obtained. This program will size this array by calling the function `size_tran_arrays`. This function will pass a store/day at a time to the functions `Delete_tran`, `Delete_cust`, `Delete_balance_group`, `Delete_total_value` and `Delete_missing_tran` to delete transactions, customers, balance groups, errors, totals and missing transactions respectively. As a processing completes for a store/day, the Store and Business Date should be written out to the purged file. If errors caused the store/day to be only partially completed, this should be noted in this log as well. If no store/days at all were purged, this message should be written out to this file.

Note: When a commit or a rollback occurs, the lock is lost and must be reobtained by calling `get_lock()`. Due to the limited size of the rollback segment, we commit at the end of each of the functions below. Thus `get_lock()` must be called after the commit statement.

`Size_tran_arrays` – This function will size the arrays needed for the deletes.

`Delete_tran` – This function will call several functions to delete all transaction level information from the database. This function will pass the array of store/days it receives to the following functions: `Delete_tran_export`, `Delete_tran_rev`, `Delete_tran_detail` and `Delete_tran_head`.

`Delete_tran_export` – This function will delete from the `sa_exported` and `sa_exported_rev` for the transactions with expired store/days.

`Delete_tran_error` – This function will delete from the `sa_error_rev` for the transactions with expired store/days.

`Delete_tran_rev` – This function will delete from the `sa_tran_tax_rev`, `sa_tran_disc_rev`, `sa_tran_item_rev`, `sa_tran_tender_rev`, and `sa_tran_head_rev` tables for transactions with expired store/days.

`Delete_tran_detail` – This function will delete from the `sa_tran_tax`, `sa_tran_disc`, `sa_tran_item` and `sa_tran_tender` tables for transactions with expired store/days.

`Delete_cust` – This function will delete from the `sa_cust_attr` and `sa_customer` tables all records related to transactions with expired store/days.

`Delete_tran_comments` – This function will delete records from the `sa_comments` table that are related to transactions with expired store/days. The `comment_type` will be 'SATR' for the transaction-level comments.

`Delete_error` – This function will delete from the `sa_error` and `sa_store_error` tables for expired store/days.

`Delete_tran_head` – This function will delete from the `sa_tran_head` table for transactions with expired store/days.

`Delete_total_value` – This function will delete from the `sa_total`, `sa_pos_value`, `sa_pos_value_wksht`, `sa_sys_value`, `sa_sys_value_wksht`, `sa_store_value`, `sa_hq_value`, `sa_error_wksht`, `sa_comments` where `comment_type` is 'SATL' and `sa_total_value_rev` tables for expired store/days.

`Delete_balance_group` – This function will delete from the `sa_comments` where `comment_type` is 'SABG' and `sa_balance_group` table for expired store/days.

`Delete_missing_tran` – This function will delete from the `sa_missing_tran` table for expired store/days.

`Delete_imp_exp_log` – This function will delete from the `sa_import_log` and `sa_export_log` tables.

Delete_store_bank_ach – This function will delete from the sa_store_ach and sa_bank_ach tables.

Delete_store_day_comments – This function will delete records from the sa_comments table that are related to expired store/days. The comment_type will be 'SASD' for the transaction-level comments.

Delete_escheat_totals – This function will delete from the sa_escheat_total and sa_escheat_voucher tables for expired store/days.

Delete_store_day_info – This function will delete from the sa_store_day table. We do not attempt to get a lock at the end of this function.

Delete_write_locks – This function will delete from the sa_store_day_write_lock table. We do not attempt to get a lock at the end of this function.

Free_tran_array– This function will free up the dynamically allocated memory from size_tran_array.

Delete_yearly_data – Three types of information are deleted in this function : flash sales information, reference information about company and store close dates and vouchers. Since the next 3 functions are not data-dependent, different threads can call them.

Delete_flash_sales – This function will delete records from the sa_flash_sales table for days that are (two years plus days_before_purge) older than the period.vdate.

Delete_voucher – This function will delete from the sa_voucher table. Voucher deletes are based on the sa_voucher.red_date or the escheat date. If the voucher has been redeemed, escheated or taken as an income adjustment, it should be deleted when the redeemed date or the escheated date is earlier than the period.vdate – sa_voucher_options.purge_no_days for the corresponding voucher type (tender_type_id). If the voucher is still in Assigned or Issued status, it has not yet been redeemed or escheated, and therefore it should not be deleted.

Final – This function should call restart_close to appropriately bookmark the program's status, close the purged file and should free any memory used.

Scheduling Considerations

This program should be run as the last program in the batch schedule. It can be run as part of the daily or monthly ReSA schedules.

Restart/Recovery

Restart/recovery is implicit in purge programs. The program only needs to be run again to restart appropriately. However, simple restart/recovery should be implemented to bookmark the status of the program.

Performance Considerations

Array Processing: As noted in the function level descriptions, as much processing as possible is done in arrays. Each driving cursor should fetch into an array, and then the same array should be used to delete from the database.

Row_id: Row_id should be selected in the driving cursors wherever possible and used during deletion as it is considerably more efficient than deleting by primary key.

Threading: This program will be threaded by store. Many of the tables to be deleted from have a store column, making the threading simple. Other tables do not have store, but instead have store_day_seq_no. To thread by store, the store associated with a store_day_seq_no will be used.

Performance: Performance is of the utmost importance to this program. In a production environment, this program may have to delete upwards of 20 million records a day. Performance should be tested against a data set of that size.

[sarules.pc]

Design Overview

Evaluating rules is integral to the sales auditing process. Rules make the comparisons between data from various sources. These comparisons find data errors that could be the result of either honest mistakes or fraud. Finding these mistakes during the sales auditing process prevents these errors from being passed on to merchandising and data warehouse systems.

Rules in ReSA are dynamic. Rules are not predefined in the system – retailers have the ability to define through the online Rule Definition Wizard. The rule definition process produces a PL/SQL package that evaluates the rule and inserts an error into the error table if the rule condition is met. This batch program is simply a shell that calls these PL/SQL packages from batch during the data import/polling process.

Scheduling Constraints

Processing Cycle:	This program should be run during each data import/polling.
Pre-Processing:	satotals.pc
Post-Processing:	Optional, depending on export scheduling.
Threading Schema:	This program runs against a single store at a time.

Restart Recovery

This program is designed to run multiple times a day, and therefore does not use restart-recovery.

Program Flow/Technical Overview

N/A

Shared Modules

SA_AUDIT_RULES_SQL.PROCESS_AUDIT_RULES – This PL/SQL function accepts a store_day_seq_no as input and evaluates all of the rules defined for that store/day combination. It also writes any errors produced by the rules to the error tables.

Function Level Description

Process() – this function contains the driving cursor. This cursor selects all of the store_day_seq_nos for the store inputted to the program that have a status of 'totaled'.

ProcessStoreDayTotals() - For each of these store_day_seq_nos selected in process, the program calls

SA_AUDIT_RULES_SQL.PROCESS_AUDIT_RULES to evaluate the rules and insert any errors into the error table.

I/O Specification

This program does not accept an input file, but does accept a store number on the command line.

[sastdycr.pc]

Design Overview

This batch program will create store/day, import log and export log records. The program should be run prior to uploading the data for a given store/day.

Scheduling Constraints

Processing Cycle: Four - just before datsys.

Threading Scheme: N/A

Restart Recovery

This program must be restartable. The logical unit of work is store. The program will be threaded by store.

Shared Modules

create_store_day_info () – This function will insert into the sa_store_day, sa_import_log, sa_flash_sales and sa_export_log tables.

Function Level Description

Init() – This function will call restart_init and get the vdate (using a cursor). The vdate +1 will be the 'action date' for which store/day records are created. The user may also pass in a date as an input parameter to the module. If this is done, the passed date will be used in place of vdate + 1.

Process() - This function will contain the driving cursor. The driving cursor should select all of the stores that are open today that do not already have store/days created for them. Three table hold store openings and closings. The company_closed table holds company wide closings. If there is a company wide closing, the company_closed_excep table holds any stores that are exceptions to the company closing (i.e. these stores are open). The location_closed table holds individual location closings. The function will call the size_array function to appropriately size the array of open stores. The array of open stores, the action date and the size of the array will be passed into the library function create_store_day_info().

Size_array() – This function will size the array that holds the open stores.

Force_commit() – This function will commit records to the database by calling restart_commit().

Final() - This function will call restart_close.

I/O Specification

N/A

[satotals.pc]

Design Overview

Totaling is integral to the sales auditing process. Totaling provides the values against which auditors can compare receipts. These comparisons find data errors that could be the result of either honest mistakes or fraud. Finding these mistakes during the sales auditing process prevents these errors from being passed on to merchandising and data warehouse systems. Totaling also provides quick access to other numeric figures about the days sales transactions.

Totaling in ReSA is dynamic. Totals are not predefined in the system – retailers have the ability to define through the online Total Definition Wizard. The total definition process produces a PL/SQL package that calculates a value for the total. This batch program is simply a shell that calls these PL/SQL packages from batch during the data import/polling process.

Scheduling Constraints

Processing Cycle: This program should be run during each data import/polling.

Scheduling Diagram: The ReSA data import/polling process involves multiple programs

Pre-Processing: saimptlog_finish.pc

Post-Processing: sarules.pc

Threading Schema: This program runs against a single store at a time.

Restart Recovery

This program is designed to run multiple times a day, and therefore does not use restart-recovery.

Program Flow/Technical Overview

N/A

Shared Modules

SA_TOTAL_SQL.PROCESS_CALC_TOTALS – This PL/SQL function accepts a store_day_seq_no as input and calculates all totals for that store/day combination. It also writes these calculated values to the appropriate values tables.

Function Level Description

Process() – this function contains the driving cursor. This cursor selects all of the store_day_seq_nos for the store inputted to the program that have a status of 'retotaling/reauditing required'.

ProcessStoreDayTotals() - For each of these store_day_seq_nos selected in process, the program calls SA_TOTAL_SQL.PROCESS_CALC_TOTALS to calculate the total values and insert them into the total values tables.

I/O Specification

This program does not accept an input file, but does accept a store number on the command line.

Retek Supplier Cost Changes Update [sccext]

Design Overview

The Supplier Cost Change Extract (sccext) module selects supplier cost change records, which are set to go into effect tomorrow and updates the RMS item/supplier table with the new cost. The item/location tables are also updated with the new cost if the cost change is a base cost change (supplier is the primary supplier for the item). In addition to the updates to the unit costs on the item/location tables when the base cost changes, records are also written for price history, supplier history and transaction-level stock ledger (if using standard cost). The costs on approved orders may also be updated if the recalculate order indicator is set to Yes for the item/supplier combination. The Pre/Post program for multi-threadable programs should be run after this program to update the status of the cost change to Extracted.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
COST_SUSP_SUP_DETAIL	No	Yes	No	No	No
COST_SUSP_SUP_HEAD	No	Yes	No	No	No
DEAL_CALC_QUEUE	No	Yes	Yes	Yes	No
DEAL_SKU_TEMP	No	No	Yes	No	No
DESC_LOOK	No	Yes	No	No	No
PACKHEAD	No	Yes	No	No	No
V_PACKSKU_QTY	No	Yes	No	No	No
PERIOD	No	Yes	No	No	No
PRICE_HIST	No	No	Yes	No	No
RAG_SKUS	No	Yes	No	Yes	No
RAG_SKUS_WH	No	Yes	No	Yes	No
RAG_STYLE	No	Yes	No	Yes	No
RAG_STYLE_WH	No	Yes	No	Yes	No
ITEM_SUPPLIER	No	Yes	No	Yes	No
SUPS	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
WIN_SKUS	No	Yes	No	Yes	No
WIN_WH	No	Yes	No	Yes	No
V_RESTART_COST_CHG	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Ad Hoc (run as required since not impacted by inventory position)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: sccext_post() – records in cost_susp_sup_head are updated to set status = 'E' where status = 'A' and active_date = (select vdate + 1 from period);

Threading Scheme: COST_CHANGE (V_restart_cost_chg)

Restart Recovery

```
EXEC SQL DECLARE c_head CURSOR FOR

SELECT h.cost_change,

       h.reason,

       TO_CHAR(h.active_date, 'DDMMYYYY'),

       h.status,

       d.supplier,

       d.sku,

       NVL(d.unit_cost,0),

       ';' || TO_CHAR(h.cost_change) ||

       ';' || TO_CHAR(d.supplier) ||

       ';' || TO_CHAR(d.sku)

FROM v_restart_cost_chg rv,

     cost_susp_sup_detail d,

     cost_susp_sup_head h

WHERE h.status = 'A'

      AND h.active_date = to_date(:tomorrow, 'DDMMYYYY')

      AND d.cost_change = h.cost_change
```

```
AND rv.driver_value = h.cost_change  
AND rv.driver_name = :ora_restart_driver_name  
AND rv.thread_val = :ora_restart_thread_val  
AND rv.num_threads = :ora_restart_num_threads  
AND (h.cost_change > NVL(:ora_restart_cost_chg, -999) OR  
      (h.cost_change = :ora_restart_cost_chg AND  
        (d.supplier > :ora_restart_supplier OR  
          (d.supplier = :ora_restart_supplier AND  
            (d.sku > :ora_restart_sku))))))  
ORDER BY h.cost_change, d.supplier, d.sku;
```

Program Flow

N/A

Shared Modules

CURRENCY_SQL.CONVERT - use this function to convert values to local and primary currency. Never call this package if the convert from currency code and convert to currency code are the same. Use the active for the cost_change as the date parameter for the conversion.

CURRENCY_LOCATION_SQL.GET_CURR_LOC - use this function to select the currency code for the supplier, as well as the current warehouse in the win_check(), rag_check(), and pack_check() functions.

SYSTEM_VALIDATE_SQL.CURRENCY_CODE - use this function to select the primary currency

UPDATE_BASE_COST – updates base cost for item if primary supplier has changed

PRICING_ATTRIB_SQL.BUILD_PACK_COST – This function will build the supplier cost for a buyer defined pack item by adding together the cost of all of its component items.

DEAL_SQL.INSERT_DEAL_SKU_TEMP – This function will insert a new record into the 'deal_sku_temp' table.

Function Level Description

get_primary_curr() - calls the package system_validate_sql to select the primary currency.

get_primary_supp_ind() -

get_supplier_curr() - calls the package currency_location_sql.get_curr_loc to retrieve the supplier's currency.

update_b_cost() – calls update_base_cost procedure

convert_cost_to_prim() - function to convert the unit_cost which is in supplier currency to the primary currency which was selected in the previous function. Use the convert_curr_sql package.

process() – main processing function

while(1) – main loop

```
{  
    if (get_primary_supp_ind() < 0) return(-1);  
  
    if (get_supplier_curr() < 0) return(-1);  
  
    if (get_primary_curr() < 0) return(-1);  
  
    if (convert_cost_to_prim() < 0) return(-1);  
  
    if (update_sku_supplier() < 0) return(-1);  
  
    if (check_for_pack() < 0) return(-1);  
  
    if (MATCH(primary_supp_ind.arr, "Y"))  
  
        if (update_b_cost() < 0 ) return(-1);
```

...

win_check() - when updating win_skus use the primary converted unit_cost from the convert_cost_to_prim() function.

win_check()

{

while (1)

{

Fetch

--error handling

Call currency_location_sql.get_curr_loc to select the currency code for the current warehouse.

--error handling

convert unit_cost from supplier currency to local currency

--error handling

cost_chg = (old_cost - local converted unit_cost)

insert_tran()

price_hist(sku)

convert cost_chg from local currency to primary currency

this is because total_cost_chg is inserted into sup_data which needs to be in primary currency

--error handling

total_cost_chg += (cost_chg * stock_qty)

stock_qty = 0

```
update win_wh
    set unit_cost = local converted unit_cost from above
where win_wh.sku = :sku
    and win_wh.wh = :mkd_wh
--error handling
}
insert_sup_data()
price_hist_base_cost(sku)
return(0);
}
```

rag_check() - when updating rag_style and rag_skus table use the primary converted unit_cost from the convert_cost_to_prim() function. In the first while loop convert unit_cost to local currency from supplier currency and use this new variable/value in the price_hist function.

```
while (1)
{
    Fetch c_skus_wh
    -- error handling
    call currency_location_sql.get_curr_loc to select the currency code for
the current warehouse
    convert unit_cost from supplier currency to local currency
    --error handling
    cost_chg = old_cost - unit_cost

    insert_tran
    price_hist(sku)
```

convert cost_chg from local currency to primary currency

this is because total_cost_chg is inserted into sup_data which needs to be in primary currency

```
total_cost_chg += (cost_chg * stock_qty);
```

move the update to rag_style_wh to here

move the update to rag_skus_wh to here

for both updates use the variable that is holding the local unit_cost

```
stock_qty = 0
```

```
} /* end while */
```

```
close c_skus_wh
```

check_for_pack() – The unit cost for buyer-defined pack items is built from the cost of the component items of the pack. Therefore, if the cost of a component item of a buyer-defined pack item is changed, the cost of the pack must change as well. The cost is built in the function pricing_attrb_sql.build_pack_cost

```
check_for_pack()
```

```
{
```

```
--no changes
```

```
while (1)
```

```
{
```

Fetch (any buyer-defined pack items which contain the item with a cost change)

```
--error handling
```

Call pricing_attrb_sql.build_pack_cost to build the new cost for the


```

pack.

--error handling

update item_supplier

    set unit_cost = new_unit_cost from build_pack_cost

where item = pack_no

    and supplier = :supplier

--error handling

}

close c_pack

return(0);

}

```

price_hist_base_cost(int item) - when inserting into price_hist a base cost record both unit_cost and base_retail must be in primary currency. base_retail will already be in primary, however unit_cost is in supplier currency

price_hist(int item) - when inserting into price_hist a price change for a location both the unit_cost and unit_retail must be in local currency. unit_retail should already be the local retail but the unit_cost variable will be in supplier currency.

get_pack_type() – retrieves a pack’s type from PACKHEAD where pack_no = sku.

insert_dst() – finds all orders that have the cost change item/supplier/country on them. For each of these orders, a new record is inserted into the DEAL_CALC_QUEUE table if it is not already on the table. If the order is already on the DEAL_CALC_QUEUE table, the ‘recalc_all_ind’ field is updated to ‘Y’.

order_exists(char * check_this) – checks to see whether or not the passed order is in the DEAL_CALC_QUEUE table.

I/O Specification

N/A

Technical Issues

N/A

Store Ship Schedule Purge [schedprg]

Design Overview

This program will purges all store_ship_date, company_closed, company_close_excep, and location_closed records whose dates are outside of the system options which control the retention period. The number of months of retention for the store ship schedule and the number of months of retention for the location close schedules is kept on the system_options table. This program should be run monthly.

The following database tables are accessed by this program:

TABLE	SELECT	INSERT	UPDATE	DELETE
SYSTEM_OPTIONS	Yes	No	No	No
STORE_SHIP_DATE	No	No	No	Yes
COMPANY_CLOSED	No	No	No	Yes
COMPANY_CLOSED_EXCEP	No	No	No	Yes
LOCATION_CLOSED	No	No	No	Yes

Scheduling Constraints

Processing Cycle: Ad Hoc

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

This program will use the commit_max_ctr on the restart_control table to periodically commit SQL delete operations.

Program Flow

N/A

Shared Modules

- Restart_init() - initialize restart recovery tables
- Restart_close() - commit changes and update restart recovery tables

Function Level Description

Init()

This function calls restart_init function to initialize the restart_recovery process. The g_l_restart_max_counter is assigned by the restart_init function from the max_commit_ctr on the restart_control table. This counter is used in the process function to perform periodic commits. This function also queries the system_options table for the ship_sched_history_mths and the loc_close_hist_months values. Also the period table for the current vdate.

Process()

This function deletes store_ship_date records that are older than the number of months specified by ship_sched_history_mths. Also, company_closed, company_closed_excep, and location_closed records that are older than the loc_close_hist_months variable are deleted. A while loop is used to force periodic commits of the deletes based on the value of g_l_restart_max_counter. This is done to ensure that rollback segments will not be exceeded in the event of considerable volume

Final()

This function calls the restart close function to update the restart recovery tables with the final program status (completed if finished successfully, aborted if handled exceptions occur).

Location Security Rebuild [slocrbld]

Design Overview

The security features being added to RMS will be maintained in the batch cycle. With each run, the changes made to the data in RMS will be brought under the security features of RMS through the running of 3 batch programs. Slocrbld.pc will handle the maintenance for the location security data. Locations will have different update/select attributes for a given user for any of a number of different functional areas like 'Pricing' or 'Clearances'. For each run, the program will use the security data defined for the user/group/functional area/organizational level to define whether a user can select or update every single location covered by the defined rules. The functional document describes the architecture of the security features and how it works. Rules that have a smaller scope overwrite those with a broader scope. For example, a user is assigned to two groups -- one of the groups has no update capability for a given region, while the other group allows updating for a specific store within that region. Which applies? The rule applying to the lowest level in the organization hierarchy is the rule granting the update capability for the store. Therefore, for the one store in the region, updates will be allowed. For the rest of the locations in the region, no updating will be allowed. In addition, if there are conflicting security definitions at the same hierarchy level because a user is associated with more than one group, the user is, as expected, granted the capability.

To accomplish this task as efficiently as possible, the program should be built as follows. It will be multi-threaded by store/warehouse, and use restart_recovery. In the Init routine, an array which will closely resemble the final destination security table, will be sized to handle all the locations in the particular thread running. This array will be loaded with all the locations and used repeatedly for every user/functional area combination. There will be an additional indicator (in addition to the select/update indicators) that will keep track of which locations have a rule affecting them and have therefore been "touched". Each rule will affect certain locations in the array and their attributes may be changed multiple times. When they are changed, this indicator will be raised. After all the rules are processed for a given user/functional area, the data in the array that has the "touched" indicator raised will be written out to sec_user_loc_matrix table and its indicator reset. This cycle will be repeated until all users and functional areas are exhausted.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
SEC_USER_GROUP	No	Yes	No	No	No
SEC_GROUP_LOC_MATRIX	No	Yes	No	No	No
V_RESTART_STORE_WH	No	Yes	No	No	No
REGION	No	Yes	No	No	No
DISTRICT	No	Yes	No	No	No
STORE	No	Yes	No	No	No
WH	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No
SEC_USER_LOC_MATRIX	No	No	Yes	No	No

Scheduling Constraints

Processing Cycle: Daily

Scheduling Diagram: Must run batch program prepost.pc with parameters slocrbld pre , slocrbld.pc and prepost.pc with parameters slocrbld post in series.

Pre-Processing: Prepost with parameters: slocrbld pre

Post-Processing: Prepost with parameters: slocrbld post

Threading Scheme: Store/Warehouse

Restart Recovery

The logical unit of work for location security rebuild will be the user-functional area (column_code). Restart/recovery will be based on the user-functional area. The restart commit counter will need to be carefully determined by each client according to the number of store/warehouse combinations that will be affected by the location security rebuild. Large location security rebuilds with thousands of stores/warehouses need smaller commit counters to avoid reprocessing large amounts of data in the event of program failure. Small location security rebuilds with small amount of stores/warehouses can have much larger commit counters since fewer rows will be inserted into the database each time for one user-functional area.

Shared Modules

N/A

Driving Cursor:

```
SELECT distinct u.user_id,
               l.column_code,
               l.region,
               l.district,
               l.store,
               l.wh,
               l.select_ind,
               l.update_ind
FROM sec_user_group u,
     sec_group_loc_matrix l,
     district d,
     store s,
     v_restart_store_wh v
WHERE u.group_id = l.group_id
     AND ((l.region = d.region
           AND l.district is NULL
           AND d.district = s.district
           AND v.driver_value = s.store)
          OR (l.district = s.district
              AND l.store is NULL
              AND v.driver_value = s.store)
          OR (l.store = v.driver_value)
          OR v.driver_value = l.wh)
```

```
AND v.num_threads = :pi_restart_num_threads  
AND v.thread_val = :pi_restart_thread_val  
AND (u.user_id > NVL(:ps_restart_user, '-999')  
     OR (u.user_id = :ps_restart_user  
         AND l.column_code > :ps_restart_column_code))  
ORDER BY u.user_id, l.column_code, l.region desc, l.district desc, l.store desc,  
         l.wh desc;
```

Function Level Description

Main

Init()

Check SYSTEM_VARIABLES.update_loc_sec_ind. If the indicator is not set then the program exit normally without further processing.

Call retek_init() to get restart-recover variables.

Get_total_locs()

Get total stores and warehouses in the current thread.

Size_loc_array()

Size location array based on the number of locations in the current thread. The location array includes region, district, loc, loc_type, select_ind, update_ind and touched columns.

Size_loc_insert_array()

Size location insert array based on the maximum commit count set in restart_control table. This array will be used for holding the exact data that is going to be

insert into the sec_user_loc_matrix table. The location insert array includes user_id, column_code, loc, loc_type, select_ind and update_ind columns.

Load_loc_array()

Load all the stores/warehouses including organizational info in the current thread to the location array.

Process()

The driving cursor is ordered to return records defining rules for entire regions first, and then those for districts, and on down. The records are processed in that order. That is to say, first work with the region level rules, then move to the more specific rules so that the rules with the smaller scope take priority over the higher level rules. Warehouses are not included in the hierarchy and Each warehouse translates into a rule to be written out.

Call `size_rule_array()` to allocate memory for arrays that store security rules.

Open the driving cursor in a while loop. Fetch the data into rule array.

Call `set_null_to_field()` to set fields to null when those fields' indicators are

-1 in the rule array.

Check if this is a second array fetch or greater, if yes, call `process_record()` to process the last record in last array fetch and the first record in current array fetch.

Open a for loop

Call `process_record()` to process the current and last record.

End of for loop

Copy the last record in the current array fetch to last rule array. Since the last record of an array fetch hasn't been processed until compared to the first record of the next array fetch. However, with each new array fetch, the last record of the previous array fetch is overwritten. Thus here it needs to be copied.

End of while loop.

`Size_rule_array()`

This function allocates memory for arrays that store security rules based on the maximum commit count set in table `restart_control` table. The rule array includes `user_id`, `column_code`, `region`, `region_ind`, `district`, `district_ind`, `store`, `store_ind`, `wh`, `wh_ind`, `select_ind` and `update_ind`.

`Set_null_to_field()`

This function loops through all the records in rule array and set a field to null when the field's indicator is -1.

`Process_record()`

This function does the majority of the processing. The data from the driving cursor is ordered by region and district. Therefore, all rules for a particular organizational hierarchy will be grouped together and processed so that a single security rule will be decided for that particular hierarchy. When multiple records do occur at the same level, the logical OR will be used to determine whether to grant update/select privileges.

Compare the user/functional area of the current record and the last record :

If it isn't new compare the hierarchy/warehouse of the current record and the last record:

If it isn't new, call logical_or_indicators() to update the current record's select and update indicators according to the logical 'OR' between the current and last records' indicators.

If it is new, call update_array() to blow security rule down to the store/warehouse level according to the last record rule.

If it is new:

Call update_array() to blow security rule down to the store/warehouse level according to the last record rule.

Call write_array() to insert the security rules of last record's user/functional area (down to store/warehouse level) to table sec_user_loc_matrix.

Call retek_force_commit() to commit the output to database and set book mark in the restart_bookmark table.

Logical_or_indicators()

This function updates the input current record's select and update indicators according to the logical 'OR' between the input current and last records' indicators. For example, if the current record's select indicator is 'N', the last record's select indicator is 'Y', then the current record's select indicator is updated to 'Y'; If the current record's select indicator is 'N', the last record's select indicator is 'N', then the current record's select indicator is kept untouched('N'). If the current record's select indicator is 'Y', no matter what last record's select indicator is, the current record's select indicator is kept untouched('Y'). So does update indicator.

Update_array()

This function updates the location array according to the input security rule.

If the input rule is a region level security rule, then loop through the location array, for all the stores within the region, set the select_inds and update_inds equal to the input rule's select_ind and update_ind, respectively. Set touched column of each processed row to 'Y'.

If the input rule is a district level security rule, then loop through the location array, for all the stores within the district, set the select_inds and update_inds equal to the input rule's select_ind and update_ind, respectively. Set touched column of each processed row to 'Y'.

If the input rule is a store level security rule, then loop through the location array, set the select_ind and update_ind of the store equal to the input rule's select_ind and update_ind, respectively. Set touched column of each processed row to 'Y'.

If the input rule is a warehouse level security rule, then loop through the location array, set the select_ind and update_ind of the warehouse equal to the input rule's select_ind and update_ind, respectively. Set touched column of each processed row to 'Y'.

Write_array()

This function writes out rows with touched = 'Y' in the location array to sec_user_loc_matrix table.

Create a loop through the location array

Check the location insert array, if it is full, call insert_array_to_table().

If the touched column of the row is 'Y', copy the row to location insert array. Set the touched column of the row to 'N'.

If the location insert array is not empty, call insert_array_to_table().

Insert_array_to_table()

This function uses array insert to insert the contents in the location insert array to table sec_user_loc_matrix.

final():

restart/recovery close

I/O Specification

N/A

Technical Issues

N/A

[soutdnld.pc]

Design Overview

Retek Demand Forecasting (RDF) requires notification when an item/store's stock on hand is at zero or below. This program will loop through the item/store tables and output any item/store combination that has a stock out condition to an output file. This output file will then be sent to RDF.

The logical unit of work (LUW) for this program is sku/store. Sku/store is a unique LUW.

Stored Procedures/Shared Modules (Maintainability)

N/A

Input Specifications

This program needs to output three fields: date, sku, and store to RDF. The fields should be sent for each store/sku combination that has a stock-on-hand less than or equal to zero. The date sent will always be the vdate from the PERIOD table. The sku and store come from either the WIN_STORE table for staple items or the RAG_SKUS_ST table for staple items. This program will look at only staple or fashion skus (no packs). All skus must be forecastable to be considered by this program. The forecastable indicator is held on the WIN_SKUS table for staple items or the RAG_STYLE table for fashion items.

RDF requires that the output files generated by this program be grouped by domain number. To accommodate this requirement, soutdnld.pc should be threaded by domain. Since threads are determined by the value of the domain ID, the restart_program_status table should contain a row for each domain ID. The thread value of the domain ID should be used as the thread value on this table. The total number of domains/number of threads should be equal to the number of rows on the restart_program_status table. This value must be entered into the restart_control table num_threads field. Note that anytime a new domain is created that an additional row should be added to the restart_program_status table with the thread value equal to the domain ID and the restart_control table num_threads field must be incremented to equal the total number of domains.

Domains can be held in RMS at the dept, class, or the subclass level. The SYSTEM_OPTIONS.DOMAIN_LEVEL holds the domain level that is being used. This dictates the program will need three different driving cursors. Which one will be used depends on the SYSTEM_OPTIONS.DOMAIN_LEVEL. The cursors will all be identical except one will join to DOMAIN_DEPT, one will join to DOMAIN_CLASS, and one will join to DOMAIN_SUBCLASS.

When the SYSTEM_OPTIONS.DOMAIN_LEVEL is 'D', the following should be used as the driving cursor:

```

SELECT wst.sku,
       wst.store,
       wst.stock_on_hand
FROM win_store wst,
     win_skus wsk,
     domain_dept dd
WHERE wst.stock_on_hand <= 0
     AND wsk.forecast_ind = 'Y'
     AND wst.sku          = wsk.sku
     AND dd.dept          = wsk.dept
     AND dd.domain_id     = :pl_thead_value
     AND (wst.sku         > NVL(:ps_restart_sku, -999) OR
          (wst.sku        = :ps_restart_sku AND
           (wst.store      > :ps_restart_store)))
UNION ALL
SELECT rst.sku,
       rst.store,
       rst.stock_on_hand
FROM rag_skus_st rst,
     rag_style rsk,
     domain_dept dd
WHERE rst.stock_on_hand <= 0
     AND rsk.forecast_ind = 'Y'
     AND rst.sku          = rsk.sku
     AND dd.dept          = rsk.dept

```

```
AND dd.domain_id    = :pl_thead_value  
  
AND (rst.sku        > NVL(:ps_restart_sku, -999) OR  
  
    (rst.sku        = :ps_restart_sku AND  
  
    (rst.store      > :ps_restart_store)))  
  
ORDER BY 1,2;
```

When the SYSTEM_OPTIONS.DOMAIN_LEVEL is 'C', the following should be used as the driving cursor:

```
SELECT wst.sku,  
  
       wst.store,  
  
       wst.stock_on_hand  
  
FROM win_store wst,  
  
     win_skus wsk,  
  
     domain_class dc  
  
WHERE wst.stock_on_hand <= 0  
  
     AND wsk.forecast_ind = 'Y'  
  
     AND wst.sku        = wsk.sku  
  
     AND dc.dept        = wsk.dept  
  
     AND dc.class       = wsk.class  
  
     AND dc.domain_id   = :pl_thead_value  
  
     AND (wst.sku       > NVL(:ps_restart_sku, -999) OR  
  
         (wst.sku       = :ps_restart_sku AND  
  
         (wst.store     > :ps_restart_store)))  
  
UNION ALL  
  
SELECT rst.sku,  
  
       rst.store,  
  
       rst.stock_on_hand  
  
FROM rag_skus_st rst,
```

```

        rag_style rsk,
        domain_class dc
WHERE rst.stock_on_hand <= 0
      AND rsk.forecast_ind = 'Y'
      AND rst.sku          = rsk.sku
      AND dc.dept          = rsk.dept
      AND dc.class         = rsk.class
      AND dd.domain_id     = :pl_thead_value
      AND (rst.sku         > NVL(:ps_restart_sku, -999) OR
           (rst.sku        = :ps_restart_sku AND
            (rst.store      > :ps_restart_store)))
ORDER BY 1,2;

```

When the SYSTEM_OPTIONS.DOMAIN_LEVEL is 'S', the following should be used as the driving cursor:

```

SELECT wst.sku,
       wst.store,
       wst.stock_on_hand
FROM win_store wst,
     win_skus wsk,
     domain_subclass sc
WHERE wst.stock_on_hand <= 0
      AND wsk.forecast_ind = 'Y'
      AND wst.sku          = wsk.sku
      AND sc.dept          = wsk.dept
      AND sc.class         = wsk.class
      AND sc.subclass      = wsk.subclass
      AND sc.domain_id     = :pl_thead_value

```

```
AND (wst.sku      > NVL(:ps_restart_sku, -999) OR
     (wst.sku      = :ps_restart_sku AND
      (wst.store    > :ps_restart_store)))

UNION ALL

SELECT rst.sku,
       rst.store,
       rst.stock_on_hand
FROM rag_skus_st rst,
     rag_style rsk,
     domain_subclass sc
WHERE rst.stock_on_hand <= 0
     AND rsk.forecast_ind = 'Y'
     AND rst.sku      = rsk.sku
     AND sc.dept      = rsk.dept
     AND sc.class     = rsk.class
     AND sc.subclass  = rsk.subclass
     AND dd.domain_id = :pl_thead_value
     AND (rst.sku      > NVL(:ps_restart_sku, -999) OR
          (rst.sku      = :ps_restart_sku AND
           (rst.store    > :ps_restart_store)))

ORDER BY 1,2;
```


Output Specifications

soutnn.dat – This is the file that will be created where nn is the thread (domain) number.

Example record for a date of 20000114, a store of 1004, and a sku of 10000266.

123456781234567890123456789012345678901234567890

200001151004 10000266

Record Name	Field Name	Field Type	Default Value	Description
	Date	Varchar2(8)	Period.vdate	The date of the stockout in YYYYMMDD format.
	Store	Varchar2(20)		The store at which the sku encountered the stockout – left justified with trailing blanks.
	SKU	Varchar2(20)		The sku that encountered the stockout – left justified with trailing blanks.

Function Level Description

Main()

The standard RMS main function. Calls init(), process(), and final().

Init()

The restart recovery is initialized by calling retek_init(), and the output file is set up. The file should be named soutnn.dat, where nn is the thread number. The C_get_vdate_and_domain_level cursor is called to get the vdate from the period table and the domain_level from system_options.

Create_foremat_strings()

This function populates a format string. The format string is used when writing lost sales to the output file.

Process()

Initializes the fetch struct, then calls the create_format_strings() function. Uses the domain_level fetched in Init() to call either Process_dept(), Process_class(), or Process_subclass(). Finally calls Free_array().

Write_lost_sales()

A pointer of the fetch struct and the current number of records to print are input. The date, sku, and store are then written to the output file. The format string defined by create_format_string() is used when doing the file write.

Process_dept()

A pointer to the fetch struct is passed in. Contains the C_dept_domain_level cursor for processing when SYSTEM_OPTIONS.DOMAIN_LEVEL = 'D'. Once the C_dept_domain_level cursor is opened, a While(1) loop is entered. Inside the While(1) loop, an array fetch is performed, and a break_flag variable is set to one if a fetch returns NO_DATA_FOUND. The number of records fetched is then passed, along with the struct containing the fetched data, to the Write_lost_sales() function. After the Write_lost_sales() function is called, the break_flag is checked, and the While(1) loop is broken if the break_flag != 0.

Process_class()

A pointer to the fetch struct is passed in. Contains the C_class_domain_level cursor for processing when SYSTEM_OPTIONS.DOMAIN_LEVEL = 'C'. Once the C_class_domain_level cursor is opened, a While(1) loop is entered. Inside the While(1) loop, an array fetch is performed, and a break_flag variable is set to one if a fetch returns NO_DATA_FOUND. The number of records fetched is then passed, along with the struct containing the fetched data, to the Write_lost_sales() function. After the Write_lost_sales() function is called, the break_flag is checked, and the While(1) loop is broken if the break_flag != 0.

Process_subclass()

A pointer to the fetch struct is passed in. Contains the C_subclass_domain_level cursor for processing when SYSTEM_OPTIONS.DOMAIN_LEVEL = 'S'. Once the C_subclass_domain_level cursor is opened, a While(1) loop is entered. Inside the While(1) loop, an array fetch is performed, and a break_flag variable is set to one if a fetch returns NO_DATA_FOUND. The number of records fetched is then passed, along with the struct containing the fetched data, to the Write_lost_sales() function. After the Write_lost_sales() function is called, the break_flag is checked, and the While(1) loop is broken if the break_flag != 0.

Init_array()

The initialized fetch struct is passed in. Allocates array space for the fetch struct arrays, using calloc. The array sizes are set to the value of the COMMIT_MAX_COUNTER.

Free_array()

Frees the array space that was allocated in Init_array().

Final()

Calls Retek_close().

Scheduling Considerations

Phase 4 daily. Any processing that updates the stock levels should be completed before this program runs.

Restart/Recovery

This program should use restart recovery. The LUW for this program is each unique sku store combination.

Product Security Rebuild [sprdrbld]

Design Overview

The security features being added to RMS will be maintained in the batch cycle. With each run, the changes made to the data in RMS will be brought under the security features of RMS through the running of 3 batch programs. Sprdrbld.pc will handle the maintenance for the product security data. SKUs will have different update/select attributes for a given user for any of a number of different functional areas like 'Pricing' or 'Clearances'. For each run, the program will use the security data defined for the user/group/functional area/merchandise level to define whether a user can select or update every single SKU covered by the defined rules. The functional document describes the architecture of the security features and how it works. Rules that have a smaller scope overwrite those with a broader scope. For example, a user is assigned to two groups -- one of the groups has no update capability for a given department, while the other group allows updating for a specific class within that department. Which applies? The rule with the lowest item hierarchy in its definition is the rule granting the update capability for the class. Therefore, for every SKU in the department and in the class will be allowed to update. For the rest of the SKUs in the department, no updating will be allowed. In addition, if there are conflicting security definitions at the same hierarchy level because a user is associated with more than one group, the user is, as expected, granted the capability.

Performance is a crucial consideration for this program as it involves writing records for different functional areas at the SKU level for every user in the system. To accomplish this task as efficiently as possible, the program should be built as follows. It will be multi-threaded by department, and use restart_recovery. In the Init routine, an array which will closely resemble the final destination security table, will be sized to handle all the SKUs in the particular thread running. This array will be loaded with all the SKUs and used repeatedly for every user/functional area combination. There will be an additional indicator (in addition to the select/update indicators) that will keep track of which SKUs have a rule affecting them and have therefore been "touched". Each rule will affect certain SKUs in the array and their attributes may be changed multiple times. When they are changed, this indicator will be raised. After all the rules are processed for a given user/functional area, the data in the array that has the "touched" indicator raised will be written out to a SQL Loader file and its indicator reset. This cycle will be repeated until all users and functional areas are exhausted.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
SEC_USER_GROUP	No	Yes	No	No	No
SEC_GROUP_PROD_MATRIX	No	Yes	No	No	No
V_RESTART_DEPT	No	Yes	No	No	No
DESC_LOOK	No	Yes	No	No	No
RAG_SKUS	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: Daily

Scheduling Diagram: Must run batch program prepost.pc with parameters sprdrbld pre , sprdrbld.pc and prepost.pc with parameters sprdrbld post in series. Then use SQL load control file sprdrbld.ctl to load the output file from sprdrbld.pc to database.

Pre-Processing: Prepost with parameters: sprdrbld pre

Post-Processing: Prepost with parameters: sprdrbld post

Threading Scheme: Department

Restart Recovery

The logical unit of work for location security rebuild will be the user-functional area (column_code). Restart/recovery will be based on the user-functional area. The restart commit counter will need to be carefully determined by each client according to the number of departments that will be affected by the product security rebuild. Large product security rebuilds with thousands of styles/skus need smaller commit counters to avoid reprocessing large amounts of data in the event of program failure. Small location security rebuilds with small amount of styles/skus can have much larger commit counters since fewer rows will be inserted into the database each time for one user-functional area.

Shared Modules

N/A

Driving Cursor:

```
SELECT u.user_id,
       p.column_code,
       p.dept,
       p.class,
       p.subclass,
       p.style,
       p.sku,
       p.select_ind,
       p.update_ind
FROM sec_user_group u,
     sec_group_prod_matrix p,
     v_restart_dept v
WHERE u.group_id = p.group_id
      AND v.driver_value = p.dept
      AND v.num_threads = :pi_restart_num_threads
      AND v.thread_val = :pi_restart_thread_val
      AND (u.user_id > NVL(:ps_restart_user, '-999')
           OR (u.user_id = :ps_restart_user
               AND p.column_code > :ps_restart_column_code))
ORDER BY u.user_id, p.column_code, p.dept, p.class desc, p.subclass desc,
         p.sku desc, p.style desc;
```

Function Level Description

Main()

Init()

Check SYSTEM_VARIABLES.update_prd_sec_ind. If the indicator is not set then the program exit normally without further processing.

Call retek_init() to get restart-recover variables.

Get_total_skus()

Get total skus in the current thread.

Size_sku_array()

Size sku array based on the number of skus in the current thread. The sku array includes dept, class, subclass, style, style_ind, sku, select_ind, update_ind and touched columns.

Load_sku_array()

Load all skus in the current thread to the sku array.

Process()

The driving cursor is ordered to return records defining rules for entire department first, and then those for class, and on down. The records are processed in that order. That is to say, first work with the department level rules, then move to the more specific rules so that the rules with the smaller scope take priority over the higher level rules.

Call size_rule_array() to allocate memory for arrays that store security rules.

Open the driving cursor in a while loop. Fetch the data into rule array.

Call set_null_to_field() to set fields to null when those fields' indicators are -1 in the rule array. Check if this is a second array fetch or greater, if yes, call process_record() to process the last record in last array fetch and the first record in current array fetch.

Open a for loop

Call process_record() to process the current and last record.

End of for loop

Copy the last record in the current array fetch to last rule array. Since the last record of an array fetch hasn't been processed until compared to the first record of the next array fetch. However, with each new array fetch, the last record of the previous array fetch is overwritten. Thus here it needs to be copied.

End of while loop.

Size_rule_array()

This function allocates memory for arrays that store security rules based on the maximum commit count set in table restart_control table. The rule array includes user_id, column_code, dept, class, class_ind, subclass, subclass_ind, style, style_ind, sku, sku_ind, select_ind and update_ind.

Set_null_to_field()

This function loops through all the records in rule array and set a field to null when the field's indicator is -1.

Process_record()

This function does the majority of the processing. The data from the driving cursor is ordered by dept, class, subclass, style, and sku such that the department level rules are selected first, then the class level, etc. Also, all rules for a particular merchandise hierarchy will be grouped together and processed so that a single security rule will be decided for that particular hierarchy. When multiple records do occur at the same level, the logical OR will be used to determine whether to grant update/select privileges.

Compare the user/functional area of the current record and the last record :

If it isn't new:

Compare the hierarchy/style/sku of the current record and the last record:

If it isn't new, call logical_or_indicators() to update the current record's select and update indicators according to the logical 'OR' between the current and last records' indicators.

If it is new, call update_array() to blow security rule down to the SKU level according to the last record rule.

If it is new:

Call update_array() to blow security rule down to the SKU level according to the last record rule.

Call write_array() to output the security rules of last record's user/functional area (down to SKU level) to SQL load file.

Call retek_force_commit() to set book mark in the restart_bookmark table.

Logical_or_indicators()

This function updates the input current record's select and update indicators according to the logical 'OR' between the input current and last records' indicators. For example, if the current record's select indicator is 'N', the last record's select indicator is 'Y', then the current record's select indicator is updated to 'Y'; If the current record's select indicator is 'N', the last record's select indicator is 'N', then the current record's select indicator is kept untouched('N'). If the current record's select indicator is 'Y', no matter what last record's select indicator is, the current record's select indicator is kept untouched('Y'). So does update indicator.

Update_array()

This function updates the SKU array according to the input security rule. There are five kinds of security rules. They are department, class, subclass, style and SKU level security rules.

If the input rule is a department level security rule, then loop through the SKU array, for all the SKUs within the department, set the select_inds and update_inds equal to the input rule's select_ind and update_ind, respectively. Set touched and style_touched indicators of each processed row to 'Y'.

If the input rule is a class level security rule, then loop through the SKU array, for all the SKUs within the class, set the select_inds and update_inds equal to the input rule's select_ind and update_ind, respectively. Set touched and style_touched indicators of each processed row to 'Y'.

If the input rule is a subclass level security rule, then loop through the SKU array, for all the SKUs within the subclass, set the select_inds and update_inds equal to the input rule's select_ind and update_ind, respectively. Set touched and style_touched indicators of each processed row to 'Y'.

If the input rule is a style level security rule, then loop through the SKU array, for all the SKUs corresponding to the style, set the select_inds and update_inds equal to the input rule's select_ind and update_ind, respectively. Set touched and style_touched indicators of each processed row to 'Y'.

If the input rule is a SKU level security rule, then loop through the SKU array, set the select_ind and update_ind of the SKU equal to the input rule's select_ind and update_ind, respectively. Set touched indicator of each processed row to 'Y'.

Write_array()

This function writes out rows with touched indicator equals 'Y' in the SKU array to SQL load file.

final():

restart/recovery close

I/O Specification

Each row of the output SQL load file outputfilename_x.dat (x is current thread number) corresponds to one record row in the sec_user_prod_matrix table. The format of the output file is as follows:

Column_code;user_id;SKU;select_ind;update_ind

Example:

PPRM;JOHN;10007986;N;N

PPRC;CLINTON;10001000;Y;N

PPRM;CLINTON;10007986;Y;Y

...

Technical Issues

N/A

Stock Count Shrinkage Update [stkdlly]

Design Overview

This program processes the 'Unit & Dollar' type of stock count that the user has submitted for processing for the stock ledger. The main functions are to calculate actual shrinkage amount that will be used to correct the book stock value on the stock ledger and to calculate a shrinkage rate.

For a subclass/location that a stock count is to be processed, this program calculates the book stock value as of the date of the stock count, by using the beginning of the month stock value from MONT_DATA and summing the transaction data from DAILY_DATA table since the beginning of the month to the date of stock count. It then compares the book stock value to the actual stock value stored on STAKE_PROD_LOC table to calculate the actual shrinkage amount.

An inter-stock-count shrinkage amount (i.e. inter_stocktake_shrink_amount, stored on MONTH_DATA table), which is an shrinkage amount that has been accumulated since the last time a stock count is taken, is added to the difference between the actual stock and book stock to come up with the actual shrinkage amount. Note that depending on the system option, budget shrink indicator, is set to 'Y' or 'N', the inter-stock-count shrinkage amount had been accumulated with either the budgeted shrinkage amount or the actual shrinkage amount that had been included in the book stock calculation by the stock ledger monthly process (salmth) since the last stock count.

TABLE	INDEX	SELECTS	INSERT	UDPATE	DELETE
SYSTEM_OPTIONS	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No
DAILY_DATA	No	Yes	No	No	No
STAKE_PROD_LOC	No	Yes	No	Yes	No
STAKE_HEAD	No	Yes	No	No	No
HALF_DATA_BUDGET	No	Yes	No	No	No
MONTH_DATA	No	Yes	No	Yes	No
HALF_DATA	No	No	No	Yes	No
WEEK_DATA	No	No	No	Yes	No

Scheduling Constraints

Processing Cycle:	PHASE 3
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	DEPT

Restart Recovery

```
EXEC SQL DECLARE c_stocktake CURSOR FOR

    SELECT      s.cycle_count,

    spl.dept,

    spl.class,

    spl.subclass,

    spl.store,

    spl.wh,

    TO_CHAR(s.stocktake_date, 'DDMMYYYY'),

    TO_NUMBER(TO_CHAR(s.stocktake_date, 'DD')),

    TO_NUMBER(TO_CHAR(s.stocktake_date, 'MM')),

    TO_NUMBER(TO_CHAR(s.stocktake_date, 'YYYY')),

    NVL(spl.total_cost,0) + NVL(spl.adjust_cost,0),

    NVL(spl.total_retail,0) + NVL(spl.adjust_retail,0),

    ROWIDTOCHAR(spl.rowid),

    ';' || TO_CHAR(spl.dept) ||

    ';' || TO_CHAR(spl.store) ||

    ';' || TO_CHAR(spl.class)

FROM  stake_prod_loc spl, stake_head s, v_restart_dept rv
```

```

WHERE      s.stocktake_date >=
to_date(:fdom_date,'DDMMYYYY')

AND    s.stocktake_date <= to_date(:next_eom_date,'DDMMYYYY')

AND    spl.cycle_count = s.cycle_count

AND    spl.processed = 'P'

AND    rv.driver_name = :ora_restart_driver_name

AND    rv.driver_value = spl.dept

AND    rv.num_threads = :ora_restart_num_threads

AND    rv.thread_val = :ora_restart_thread_val

AND    (spl.dept > NVL(:ora_restart_dept, -999) OR
        (spl.dept = :ora_restart_dept AND
         (spl.class > :ora_restart_class OR
          (spl.class = :ora_restart_class AND spl.store >
           :ora_restart_store))))))

ORDER BY  spl.dept,
          spl.class,
          spl.store;

```

Program Flow

N/A

Shared Modules

STKLEDGR_ACCTING_SQL.RETAIL_METHOD_CALC: performs stock ledger stock and gross margin calculations using the retail accounting method.

STKLEDGR_ACCTING_SQL.COST_METHOD_CALC: performs stock ledger stock and gross margin calculations using the cost accounting method.

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Purge Stock Count [stkprg]

Design Overview

The function of this program is to delete records from the cycle count tables for stock takes with a STAKE_HEAD.STOCKTAKE_DATE that is less than the SYSTEM_VARIABLES.LAST_EOM_DATE. The program works through stake_head looking for out of date records and then deletes them along with their child records from the stake_sku_loc and stake_prod_loc tables.

Re-run:

If this program terminates normally, restart without recovery.

If this program terminates abnormally, restart without recovery.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
STAKE_HEAD	No	Yes	No	No	Yes
STAKE_LOCATION	No	No	No	No	Yes
STAKE_PRODUCT	No	No	No	No	Yes
STAKE_PROD_LOC	No	No	No	No	Yes
STAKE_QTY	No	No	No	No	Yes
STAKE_SKU_LOC	No	No	No	No	Yes

Scheduling Constraints

Processing Cycle: AD HOC

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

```
EXEC SQL DECLARE get_old_records CURSOR FOR

      SELECT      cycle_count

      FROM stake_head

      WHERE      ((stocktake_date + :no_days_before_purge) <

(select vdate from period))

      OR      delete_ind = 'Y';
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Stock Count Snapshots Update [stkupd]

Design Overview

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
STAKE_SKU_LOC	Yes	Yes	No	No	No
PERIOD	No	Yes	No	No	No
WIN_STORE	Yes	No	No	Yes	No
WIN_WH	Yes	No	No	Yes	No
RAG_SKUS_ST	Yes	No	No	Yes	No
RAG_SKUS_WH	Yes	No	No	Yes	No
PACKWH	No	Yes	No	No	No

Indexes: STAKE_DATE (stocktake_date),

WIN_STORE (sku, store),
 WIN_WH (sku, wh),
 RAG_SKUS_ST (style, store),
 RAG_SKUS_WH (style, wh),
 PACKWH (pack_no, wh)

Sets the stake_sku_loc snapshot_on_hand_qty to the current win_store/win_wh (or rag_skus_st/rag_skus_wh) stock on hand for each store and warehouse for which a stocktake has been scheduled for today. Also fetches snapshot_unit_retail & snapshot_unit_cost the same way.

Re_run:

If this program terminates normally/abnormally, the restart_program_status table must be updated by setting restart_flag to 'Y' for current restart_name, and schema.

Scheduling Constraints

Processing Cycle: PHASE 3

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: STORE/WH

Restart Recovery

```
EXEC SQL DECLARE c_get_sku_loc CURSOR FOR

    SELECT      stake_sku_loc.sku,

    stake_sku_loc.store,

    stake_sku_loc.wh,

    stake_sku_loc.cycle_count,

    ROWIDTOCHAR(stake_sku_loc.ROWID),

    desc_look.system_ind,

    ';' || TO_CHAR(stake_sku_loc.sku) ||

    ';' || TO_CHAR(stake_sku_loc.store) ||

    ';' || TO_CHAR(stake_sku_loc.wh)

    FROM stake_sku_loc,

    desc_look,

    stake_head,

    v_restart_store_wh rv

    WHERE      stake_sku_loc.sku = desc_look.sku

    AND  stake_sku_loc.cycle_count = stake_head.cycle_count

    AND  stake_head.stocktake_date = (select period.vdate from period)

    AND  rv.driver_value      = stake_sku_loc.store

    AND  rv.driver_name      = :ora_restart_driver_name

    AND  rv.num_threads      = :ora_restart_num_threads

    AND  rv.thread_val       = :ora_restart_thread_val

    AND (stake_sku_loc.sku >

        NVL(:ora_restart_sku, stake_sku_loc.sku - 1) OR

    (stake_sku_loc.sku = :ora_restart_sku AND
```

```
(stake_sku_loc.store > :ora_restart_store OR
(stake_sku_loc.store = : ora_restart_store AND
stake_sku_loc.wh > :ora_restart_wh)
)
)
)
ORDER BY stake_sku_loc.sku,
stake_sku_loc.store,
stake_sku_loc.wh;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Upload Stock Count Results [stkupld]

Design Overview

The purpose of this batch module is to accept cycle count details from an external system. The cycle count transactions will be compared with Retek system snapshots of stock on hand at the time of the cycle count to determine the stock and/or dollar adjustments to be made. The following common functions will be performed on each stock record read from the input file:

- get system indicator for SKU
- if record exists on STAKE_SKU_LOC then update it
- if record doesn't exist on STAKE_SKU_LOC
 - validate that SKU/location exists in system
 - insert a record into STAKE_SKU_LOC
 - insert stock take record information into STAKE_QTY
- if record is a pack - update/insert information on STAKE_SKU_LOC for all component SKUs

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
stake_qty	No	No	Yes	No	No
stake_sku_loc	No	No	Yes	Yes	No
upc_ean	No	Yes	No	No	No
rag_skus_st	No	Yes	No	No	No
win_store	No	Yes	No	No	No
rag_skus_wh	No	Yes	No	No	No
rag_skus	No	Yes	No	No	No
rag_style_st	No	Yes	No	No	No
rag_style_wh	No	Yes	No	No	No
packhead	No	Yes	No	No	No

This program reads a user-created interface file of cycle counts. Files will be unique to location and cycle count ID. All records will be validated for layout. Invalid layouts will produce fatal errors. Fields will be validated for content. Invalid contents will produce non-fatal errors. Valid records will update the `physical_count_qty` field on `STAKE_SKU_LOC` for a given SKU/location/ cycle count combination. If the SKU is a pack, component SKUs will have their component quantity added to the `pack_comp_qty` field on `STAKE_SKU_LOC`. If a SKU does not exist on `STAKE_SKU_LOC`, the SKU/location combination will be validated on the SKU/location tables and a new record will be inserted to `STAKE_SKU_LOC`.

Fatal errors will terminate file processing. Non-fatal errors will discontinue record processing and will write invalid record to a reject file.

File layout will be verified by interface library routines:

- `get_record`: validates common fields in file head record and fills structure of remaining fields that are passed from this program
- `process_dtl_ftail`: called after end-of-file is reached. Will process file trailer record by validating its layout and verifying that the file record counter is set properly.

Re-run:

If this program terminates normally, restart without recovery.

If this program terminates abnormally, restart without recovery.

Scheduling Constraints

Processing Cycle: PHASE 3 (Daily)

Scheduling Diagram: This program will probably be run at the start of the batch cycle during POS polling, or possibly at the end of the batch run if pending warehouse transactions exist. It can be scheduled to run multiple times throughout the day, as WMS or POS data becomes available.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

The logical unit of work for the stock take upload module will be a count of discrete inventory transactions. Each record will be uniquely identified by location (store or warehouse), SKU and description of inventory count location. The logical unit of work will be defined as a number of these transaction records, determined by the `commit_max_ctr` field on the `restart_control` table.

The file records will be grouped in numbers equal to the `commit_max_ctr`. After all records in a given read are processed (or rejected), the restart commit logic and restart file writing logic will be called, after which the following group of file records will be read and processed. The commit logic will save the current file pointer position in the input file and any application image information (e.g. record and reject counters) and commit all database transactions. The file writing logic will append the temporary holding files to the final output files.

The `commit_max_ctr` field should be set to prevent excessive rollback space usage and to reduce the overhead of file I/O. The recommended commit counter setting is 10,000 records (subject to change based on experimentation).

Error handling will recognize three levels of record processing: process success, non-fatal errors, and fatal errors. Item level validation will occur on all fields before table processes are initiated. If all field-level validations return successfully, inserts and updates will be allowed. If a non-fatal error is produced, the remaining fields will be validated but the record will be rejected and written to the reject file. If a fatal error is returned, file processing will end immediately. A restart will be initiated from the file pointer position saved in the `restart_bookmark` string at the time of the last commit point that was reached during file processing.

Program Flow

N/A

Shared Modules

`validate_all_numeric`: interface library function.

`validate_all_numeric_signed`: interface library function.

`valid_date`: interface library function.

Function Level Description

init():

initialize restart recovery

open input file

- file should be specified as input parameter to program

declare final output filename (used in restart_write_file logic)

open reject file (as a temporary file for restart)

- file should be specified as input parameter to program

call restart_file_init logic

assign application image array variables- line counter (g_l_rec_cnt), reject counter (g_l_rej_cnt), cycle_count, stocktake date

if fresh start (l_file_start = 0)

read file header record (get_record)

if (record type $\neq$ 'FHEAD') Fatal Error

validate file type = 'STKU'

else fseek to l_file_start location

validate cycle count id & cycle count date are valid on stake_head

process()

malloc structure for input to stake_qty to hold commit_max_ctr elements

loop - fread rows (equal to commit counter) of input file

if end of file encountered, decrement for loop counter and set end of file flag to true

for loop to process all records read

copy input detail structure elements to stake_sku_loc structure elements

validate elements (val_FDETL)

if non-fatal error occurs write detail structure to reject file (write_to_rej_file) and continue at the top of the for-loop

update stake_sku_loc

if record doesn't exist, validate that sku/location is valid

if invalid then non-fatal error -write record & continue

insert to stake_sku_loc (if display pack also insert component skus)

fill current stake_qty structure & increment valid record counter

end loop for loop to process individual records

insert structure of arrays (for valid record counter) into stake_sku_loc

restart file commit - save current input file position, and application image (cnt, cycle count & date)

restart write file function

if end of file reached then break from while loop

end outer loop to read from file

restart commit final

validate_FDETL()

if record type != FDETL then fatal file layout error

do standard string validations - if any return non-fatal error then set non-fatal error flag to true

 nullpad all fields

 left shift location description field, sku, store, qty, upc, upc_sup

 check that sku, store, qty, upc, upc_sup are all numeric

 check that upc & sku are not both filled

 check that if upc exists and upc_sup is blank, upc_sup = 00000

 place decimal in qty field

 validate location type = 'ST' or 'WH'

 validate date field

- find SKU from UPC and UPC supplement (old upc_to_sku)

if not found non-fatal error

-get sku system indicator and department

if (system indicator does not exist) set non-fatal error flag to true

ON Fatal Error

Exit Function with -1 return code

ON Non-Fatal Error

write out rejected record to the reject file using write_to_rej_file functionn, pass pointer to detail record structure, number of bytes in structure, and reject file pointer

I/O Specification

Input File

The input file should be accepted as a runtime parameter at the command line.

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	ID of current line being processed by input file.
	File Type Definition	Char(4)	STKU	Identifies file as 'Stake Upload'
	File Create Date	Date	create date	date file was written by external system
	Stocktake Date	Date	specified by external system	date inventory was counted (should be equal to stocktake_date on stake_head)
	Cycle Count ID	Number(8)	cycle count number on stake_head	Cycle count number for detail values. Assigned by Retek.
	Location Type	Char(2)	ST WH	'ST' for store or 'WH' for warehouse
	Location Number	Number(4)	specified by external system	Store or warehouse identifier
Transaction Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	specified by external system	ID of current line being processed by input file.
	Item Type	Char(3)	UPC SKU	item type will be represented as an UPC, an SKU

Record Name	Field Name	Field Type	Default Value	Description
	Item Value	Number(13)	item identifier	the id number of a SKU or UPC
	Supplement	Number(5)	supplemental identifier	used to further specify the id of an UPC item, or the pre-pack id reference
	Inventory Quantity	Number(12,4)		Number of units counted at a particular location
	Location Description	Char(30)		Description of location where item was counted
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	specified by external system	ID of current line being processed by input file.
	File Record Counter	Number(10)		Number of records/transactions processed in current file (only records between head & tail)

Reject File

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will be created by the interface library routines and the detail records will be created using the `write_to_rej_file` function. A reject line counter will be kept in the program and is required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and trailer record and the file line count will be equal to 0.

A final reject file name, a temporary reject file name, and a reject file pointer should be declared. The reject file pointer will identify the temporary reject file. This is for the purposes of restart recovery. When a commit event takes place, the `restart_write_function` should be called (passing the file pointer, the temporary name and the final name). This will append all of the information that has been written to the temp file since the last commit to the final file. Therefore, in the event of a restart, the reject file will be in synch with the input file.

Error File

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

N/A

Stock Count Stock on Hand Updates [stkvar]

Design Overview

Table	Index	Select	Insert	Update	Delete
STAKE_SKU_LOC	no	yes	no	yes	no
STAKE_CONT	no	yes	no	no	yes
STAKE_HEAD	no	yes	no	no	no
WIN_SKUS	yes	yes	no	no	no
WIN_STORE	yes	no	no	yes	no
WIN_WH	yes	no	no	yes	no
RAG_STYLE	yes	yes	no	no	no
RAG_SKUS	yes	yes	no	no	no
RAG_SKUS_ST	yes	no	no	yes	no
RAG_SKUS_WH	yes	no	no	yes	no
STAKE_PROD_LOC	yes	no	no	yes	no
DESC_LOOK	no	yes	no	no	no
PACKWH	yes	yes	no	yes	no

Indexes: WIN_SKUS (dept),
 WIN_STORE (sku, store),
 WIN_WH (sku, wh),
 RAG_STYLE (dept),
 RAG_SKUS (sku),
 RAG_SKUS_ST (sku, store),
 RAG_SKUS_WH (sku, wh),
 PACKWH (pack_no, wh),
 STAKE_PROD_LOC (dept, store, wh, data_type)

This program updates the stock on hand for all items as a result of a stock take.

The program is driven by STAKE_CONT, in conjunction with STAKE_SKU_LOC where the SKU, store, warehouse and cycle count on STAKE_SKU_LOC match those on STAKE_CONT, and where STAKE_CONT run_type = 'A' (for adjustment).

For each row retrieved from the above tables, the unit systems are processed as follows:

A WIN_STORE or WIN_WH record is updated for every SKU/store combination (RAG_SKUS_ST and RAG_SKUS_WH for fashion, PACKWH for packs). The new stock on hand = win_store.stock_on_hand - snapshot_stock_on_hand_qty (from the STAKE_SKU_LOC table) + the physical count quantity on STAKE_SKU_LOC. In addition, the pack_comp_soh field is updated on WIN_WH or RAG_SKUS_WH when a pack is processed for each component SKU in the pack.

Total cost and total retail are computed as the snapshot unit retail times the sum of the physical count quantity plus the snapshot in-transit (from the STAKE_SKU_LOC table).

STAKE_PROD_LOC total cost and total retail amounts are updated with the total cost and total retail for each department, class, subclass, store and warehouse combination that exists on the cycle count. A record for each is added. If a record already exists on the table, the total cost or retail amount value is adjusted to be the existing total cost or retail amount + the cycle count cost or retail. If no record exists, a new one is added to the table with the value of total cycle count cost or retail for the total cost or retail amount. If the stock ledger is designated not to include VAT on the SYSTEM_OPTIONS table, the total retail amount will have any VAT amount stripped from it.

Re-run: If this program terminates normally, WIN_STORE, WIN_WH, PACKWH, STAKE_QTY, RAG_SKUS_ST, RAG_SKUS_WH, STAKE_PROD_LOC_STOCK and STAKE_CONT must be recovered prior to restart. If this program terminates abnormally, restart without recovery.

The syntax for invoking this program is:

stkvar userid/pswd [report_name].

Here are some examples:

stkvar userid/pswd (it will not produce any report.)

stkvar userid/pswd any.rpt (it will produce a report, any.rpt.)

Scheduling Constraints

Processing Cycle:	PHASE 3
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	DEPT

Restart Recovery

```
EXEC SQL DECLARE c_skus CURSOR FOR

    SELECT      stake_cont.sku,
               stake_cont.store,
               stake_cont.wh,
               stake_sku_loc.snapshot_on_hand_qty,
               nvl(stake_sku_loc.snapshot_in_transit_qty, 0),
               nvl(stake_sku_loc.snapshot_unit_cost, 0),
               nvl(stake_sku_loc.snapshot_unit_retail, 0),
               stake_sku_loc.physical_count_qty,
               stake_sku_loc.pack_comp_qty,
               nvl(stake_sku_loc.dept, 0),
               nvl(stake_sku_loc.class, 0),
               nvl(stake_sku_loc.subclass, 0),
               desc_look.system_ind,
               stake_head.stocktake_date,
               stake_head.stocktake_type,
               stake_head.cycle_count,
               ';' || TO_CHAR(stake_cont.sku) ||
```

```
;' || TO_CHAR(stake_cont.store) ||  
';' || TO_CHAR(stake_cont.wh)  
  
FROM stake_head,  
  
desc_look,  
  
stake_sku_loc,  
  
stake_cont,  
  
v_restart_dept rv  
  
WHERE      stake_cont.run_type = 'A'  
  
AND  stake_cont.sku = stake_sku_loc.sku  
  
AND  stake_cont.store = stake_sku_loc.store  
  
AND  stake_cont.wh = stake_sku_loc.wh  
  
AND  stake_cont.cycle_count = stake_sku_loc.cycle_count  
  
AND  stake_head.cycle_count = stake_cont.cycle_count  
  
AND  desc_look.sku = stake_cont.sku  
  
AND  rv.driver_value  = desc_look.dept  
  
AND  rv.driver_name   = :ora_restart_driver_name  
  
AND  rv.num_threads   = :ora_restart_num_threads  
  
AND  rv.thread_val    = :ora_restart_thread_val  
  
AND  (stake_cont.sku  > NVL(:ora_restart_sku,-999) OR  
  
      (stake_cont.sku = :ora_restart_sku AND  
  
      (stake_cont.store > :ora_restart_store OR  
  
      (stake_cont.store = :ora_restart_store AND  
  
      stake_cont.wh > :ora_restart_wh)  
  
      )  
  
      )  
  
      )
```



```
ORDER BY    stake_cont.sku,  
            stake_cont.store,  
            stake_cont.wh;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Stock Count Explode to SKU/Location for Stores [stkxplst]

Design Overview

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DESC_LOOK	Yes	Yes	No	No	No
STAKE_SKU_LOC	Yes	No	Yes	No	No
STAKE_PROD_LOC	Yes	No	Yes	No	No
STAKE_LOCATION	Yes	Yes	No	No	No
STAKE_HEAD	Yes	Yes	No	No	No
STAKE_PRODUCT	Yes	Yes	No	No	No
WIN_STORE	Yes	Yes	No	No	No
RAG_SKUS_ST	Yes	Yes	No	No	No
WIN_SKU	Yes	Yes	No	No	No
RAG_SKUS	Yes	Yes	No	No	No
RAG_STYLE	Yes	Yes	No	No	No

This program inserts into stake_sku_loc and stake_prod_loc tables for all valid SKU/store and department/store combinations for unit and dollar type stock counts, based on store, department, class and subclass information stored on stake_location and stake_product. The stock count can be requested at department, class or subclass level and exploded to SKU/store from any of these levels. Inserts are made into stake_prod_loc now at subclass/store level.

Re-run:

If this program aborts and terminates normally or if this program terminates abnormally, restart after setting restart_flag = 'Y' on the restart_program_status table for current restart_name, schema, and thread_val.

Scheduling Constraints

Processing Cycle: PHASE 3

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: DEPT

Restart Recovery

```

EXEC SQL DECLARE C_product CURSOR FOR

      SELECT      h.cycle_count,

      s.dept,

      s.class,

      s.subclass,

      ';' || TO_CHAR(h.cycle_count) ||

      ';' || TO_CHAR(s.dept) ||

      ';' || TO_CHAR(s.class) ||

      ';' || TO_CHAR(s.subclass)

FROM stake_product p,

stake_head h,

subclass s,

v_restart_dept rv

WHERE      h.cycle_count = p.cycle_count

AND      h.stocktake_date =
(TO_DATE(:threshold_date,'DDMMYYYY'))

AND      h.stocktake_type = 'B'

AND      h.store_or_warehouse = 'S'

AND      h.delete_ind = 'N'

AND      s.dept = p.dept

AND      s.class = NVL(p.class,s.class)

AND      s.subclass = NVL(p.subclass,s.subclass)

AND      rv.driver_value = p.dept

AND      rv.driver_name = :ora_restart_driver_name

AND      rv.num_threads = :ora_restart_num_threads

AND      rv.thread_val = :ora_restart_thread_val

```

```
AND (h.cycle_count > NVL(:ora_restart_cycle_count, -999) OR
(h.cycle_count = :ora_restart_cycle_count AND
(s.dept > :ora_restart_dept OR
(s.dept = :ora_restart_dept AND
(s.class > :ora_restart_class OR
(s.class = :ora_restart_class AND
(s.subclass >= :ora_restart_subclass))))))
ORDER BY h.cycle_count,
s.dept,
s.class,
s.subclass;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Stock Count Explode to SKU/Location for Warehouses [stkxplwh]

Design Overview

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DESC_LOOK	Yes	Yes	No	No	No
STAKE_SKU_LOC	Yes	No	Yes	No	No
STAKE_PROD_LOC	Yes	no	Yes	No	No
STAKE_LOCATION	Yes	Yes	No	No	No
STAKE_HEAD	Yes	Yes	No	No	No
STAKE_PRODUCT	Yes	Yes	No	No	No
WIN_WH	Yes	Yes	No	No	No
RAG_SKUS_WH	Yes	Yes	No	No	No
PACKWH	Yes	Yes	No	No	No
PACKHEAD	Yes	Yes	No	No	No
WIN_SKUS	Yes	Yes	No	No	No
RAG_SKUS	Yes	Yes	No	No	No

This program inserts into the stake_sku_loc and stake_prod_loc tables for all valid SKU/warehouse and department/warehouse combinations for unit and dollar type stock counts, based on warehouse, department, class and subclass information stored on stake_location and stake_product. The stock count can be requested at department, class or subclass level and exploded to SKU/warehouse from any of these levels. Inserts are made to stake_prod_loc now at subclass/warehouse level.

Re-run:

If this program aborts and terminates normally or if this program terminates abnormally, restart after setting restart_flag = 'Y' on restart_program_status for the current restart_name, schema and thread_val.

Scheduling Constraints

Processing Cycle: PHASE 3
Scheduling Diagram: N/A
Pre-Processing: N/A
Threading Scheme: DEPT

Restart Recovery

```
EXEC SQL DECLARE c_product CURSOR FOR

    SELECT      h.cycle_count,
               s.dept,
               s.class,
               s.subclass,
               ';' || TO_CHAR(h.cycle_count) ||
               ';' || TO_CHAR(s.dept) ||
               ';' || TO_CHAR(s.class) ||
               ';' || TO_CHAR(s.subclass)
    FROM v_restart_dept rv,
         stake_product p,
         stake_head h,
         subclass s

    WHERE      p.cycle_count = h.cycle_count

              AND h.stocktake_date =
(TO_DATE(:threshold_date,'DDMMYYYY'))

              AND h.stocktake_type = 'B'

              AND h.store_or_warehouse = 'W'

              AND h.delete_ind = 'N'

              AND s.dept = p.dept
```

```
AND s.class = NVL(p.class,s.class)
AND s.subclass = NVL(p.subclass,s.subclass)
AND rv.driver_value = p.dept
AND rv.driver_name = :ora_restart_driver_name
AND rv.num_threads = :ora_restart_num_threads
AND rv.thread_val = :ora_restart_thread_val
AND (h.cycle_count > NVL(:ora_restart_cycle_count, -999) OR
      (h.cycle_count = :ora_restart_cycle_count AND
       (s.dept > :ora_restart_dept OR
        (s.dept = :ora_restart_dept AND
         (s.class > :ora_restart_class OR
          (s.class = :ora_restart_class AND
           (s.subclass >= :ora_restart_subclass)))))))
ORDER BY h.cycle_count,
         s.dept,
         s.class,
         s.subclass;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Stock Ledger Extract [stlgdnld.pc]

Design Overview

This program extracts stock ledger data in addition to on-order quantities at the department/class/subclass/location/week level. The program can extract data for a historic period or for the most current complete week. The program expects an input file that determines whether the extract is a historic extract or a weekly extract. As well, in order to calculate on-order information, the program reads from a temporary on-order table that is populated by an on-order extract program that is meant to run before this program.

The information extracted by this program includes: Sales Value, BOP Value, BOP Cost, RTV Value, Reclass In, Reclass Out, Permanent Markdown Value, Promotional Markdown Value, Clearance Markdown Value, Shrinkage Value, Received Value, Received Cost, On-order Value, On-order Cost, Transfer In Value, Transfer In Cost, Transfer Out Value, Transfer Out Cost, Sales Unit.

This information is intended to be used by the Retek Predictive Planning application.

Input Specifications

Input file:

Will have the following format:

H[date in the 'YYYYMMDD' format][date in the 'YYYYMMDD' format] or

W[from date in the 'YYYYMMDD' format][to date in the 'YYYYMMDD' format]

Ex: H1996010120000222 or W199601012000022

Driving Cursor:

The eow_date filtering will be driven by the download type input parameter, whether the extract will occur for a historic date range (H) for the most current complete week (W).

Driving cursor for current complete week:

```
SELECT w.dept,
       w.class,
       w.subclass,
       DECODE(w.store,-1,'W','S'),
       DECODE(w.store,-1,w.wh,w.store),
       w.eow_date,
       NVL(w.net_sales_retail,0),
       NVL(w.opn_stk_retail,0),
       NVL(w.opn_stk_cost,0),
       NVL(w.rtv_retail,0),
       NVL(w.reclass_in_retail,0),
       NVL(w.reclass_out_retail,0),
       NVL(w.perm_markdown_retail,0),
       NVL(w.prom_markdown_retail,0),
       NVL(w.clear_markdown_retail,0),
       NVL(w.shrinkage_retail,0),
       NVL(w.purch_retail,0),
       NVL(w.purch_cost,0),
       sum(NVL(ot.on_order_retail,0)),
       sum(NVL(ot.on_order_cost,0)),
       NVL(w.tsf_in_retail,0),
       NVL(w.tsf_in_cost,0),
       NVL(w.tsf_out_retail,0),
       NVL(w.tsf_in_cost,0),
       NVL(w.sales_units,0)
```

```

FROM week_data w,
     v_restart_dept rv,
     on_order_temp ot
WHERE w.currency_ind = 'L'
     AND w.eow_date = :ps_eow_date
     AND w.dept      = ot.dept(+)
     AND w.class     = ot.class(+)
     AND w.subclass  = ot.subclass(+)
     AND w.store     = ot.store(+)
     AND w.wh        = ot.wh(+)
     AND rv.driver_value = w.dept
     AND rv.driver_name = 'DEPT'
     AND rv.num_threads = :pi_num_threads
     AND rv.thread_val  = :pi_thread_val
     AND (w.dept > NVL(:ps_restart_dept, -999) OR
          (w.dept = :ps_restart_dept AND
           (w.class > :ps_restart_class OR
            (w.class = :ps_restart_class AND
             (w.subclass > :ps_restart_subclass))))))
GROUP BY w.dept,
         w.class,
         w.subclass,
         DECODE(w.store,-1,'W','S'),
         DECODE(w.store,-1,w.wh,w.store),
         w.eow_date,
         NVL(w.net_sales_retail,0),

```

```
NVL(w.opn_stk_retail,0),
NVL(w.opn_stk_cost,0),
NVL(w.rtv_retail,0),
NVL(w.reclass_in_retail,0),
NVL(w.reclass_out_retail,0),
NVL(w.perm_markdown_retail,0),
NVL(w.prom_markdown_retail,0),
NVL(w.clear_markdown_retail,0),
NVL(w.shrinkage_retail,0),
NVL(w.purch_retail,0),
NVL(w.purch_cost,0),
NVL(w.tsf_in_retail,0),
NVL(w.tsf_in_cost,0),
NVL(w.tsf_out_retail,0),
NVL(w.tsf_in_cost,0),
NVL(w.sales_units,0)
ORDER BY w.dept,
w.class,
w.subclass,
w.eow_date;
```

Driving cursor for historic data:

```
SELECT w.dept,
       w.class,
       w.subclass,
       DECODE(w.store,-1,'W','S'),
       DECODE(w.store,-1,w.wh,w.store),
       w.eow_date,
       NVL(w.net_sales_retail,0),
       NVL(w.opn_stk_retail,0),
       NVL(w.opn_stk_cost,0),
       NVL(w.rtv_retail,0),
       NVL(w.reclass_in_retail,0),
       NVL(w.reclass_out_retail,0),
       NVL(w.perm_markdown_retail,0),
       NVL(w.prom_markdown_retail,0),
       NVL(w.clear_markdown_retail,0),
       NVL(w.shrinkage_retail,0),
       NVL(w.purch_retail,0),
       NVL(w.purch_cost,0),
       NVL(w.tsf_in_retail,0),
       NVL(w.tsf_in_cost,0),
       NVL(w.tsf_out_retail,0),
       NVL(w.tsf_in_cost,0),
       NVL(w.sales_units,0)
FROM week_data w,
     v_restart_dept rv
```

```
WHERE w.currency_ind = 'L'

AND w.eow_date >= :is_from_date

AND w.eow_date <= :is_to_date

AND rv.driver_value = w.dept

AND rv.driver_name = 'DEPT'

AND rv.num_threads = :pi_num_threads

AND rv.thread_val = :pi_thread_val

AND (w.dept > NVL(:ps_restart_dept, -999) OR
      (w.dept = :ps_restart_dept AND
        (w.class > :ps_restart_class OR
          (w.class = :ps_restart_class AND
            (w.subclass > :ps_restart_subclass))))))

ORDER BY w.dept,
         w.class,
         w.subclass,
         w.eow_date;
```

Output Specifications

Output Files

The file outputted will be named stckldgr.dat.

Output File Format:

Record Name	Field Name	Field Type	Default Value	Description
	Department	Char(4)		Department of sales/on-order data
	Class	Char(4)		Class of sales/on-order data
	Subclass	Char(4)		Subclass of sales/on-order data
	Store	Char(4)		Store of sales/on-order data
	EOW Date	Char(8)		End of week date of week for which data was derived
	Sales Value	Number(21)		Total sales for subclass/store/week (retail)
	BOP Value	Number(21)		Beginning of Period value (retail)
	BOP Cost	Number(21)		Beginning of Period value (cost)
	RTV Value	Number(21)		Return to Vendor value (retail)
	Reclass In	Number(21)		Reclass In value (retail)
	Reclass Out	Number(21)		Reclass Out value (retail)
	Permanent Markdown Value	Number(21)		Permanent markdown value (retail)
	Promotional Markdown Value	Number(21)		Promotion markdown value (retail)
	Clearance Markdown Value	Number(21)		Clearance markdown value (retail)
	Shrinkage Value	Number(21)		Shrinkage value (retail)
	Received Value	Number(21)		Received value (retail)
	Received Cost	Number(21)		Received value (cost)
	On-order Value	Number(21)		On order value (retail) only in current week
	On-order Cost	Number(21)		On order value (cost) only in current week
	Transfer In Value	Number(21)		Transfer In value (retail)

Record Name	Field Name	Field Type	Default Value	Description
	Transfer In Cost	Number(21)		Transfer In value (cost)
	Transfer Out Value	Number(21)		Transfer Out value (retail)
	Transfer Out Cost	Number(21)		Transfer Out value (cost)
	Sales Unit	Number(13)		Sales Units

Function Level Description

`main()`:

The standard Retek `main()` function. Calls `init()`, `process()`, and `final()`.

`init()`:

Initialize restart recovery by calling `retex_init()` and set up the output file.

`format_buffer()`:

Formats the string which will be used to write to the output file.

`process()`:

Will read the input file and Calls either `fetch_w_process` or `fetch_h_process` according to the input value from command line. Prior to calling any process function, it calls `get_multi_currency_ind` so they can be passed to process functions.

`fetch_w_process()`

The bulk work of traditional process function is done here. This function will employ the driving cursor specifically coded to fetch the most current end of week information from `WEEK_DATA` table and the corresponding values for `on_order_retail` and `on_order_cost` from `ON_ORDER_TEMP` tables. This function will also loop through records returned by the driving cursor and write to the output file by calling `write_file()`. Also within the for loop it calls the conversion function if the `multi_currency_ind = 'Y'`.

`fetch_h_process()`

Same as `fetch_w_process`, except this function will employ the driving cursor for historic data and call `historic_eow_date()`.

`get_eow_date()`

Fetches the most current end of week date from `SYSTEM_VARIABLES.LAST_EOW_DATE_UNIT`.

`get_multi_currency_ind()`

Fetches multi-currency indicator value from `SYSTEM_OPTIONS`.

`historic_eow_date()`

From `_date` and `to_date` from command line are passed to determine the appropriate end of week date for input values using `DATES_SQL.GET_EOW_DATE`, a SQL function.

`write_file()`

This function will employ `rtk_print` to write the information fetched from driving cursor to the output file.

`conversion()`

This function will call a C function, `CONVERT_CURRENCY`, from `utils.h` and convert all the amount values to primary currency if the multi-currency-indicator from `system_options` is 'Y'. This function will be called within the for loop of both `fetch_process` functions.

`size_array()`

Dynamically allocated memory to arrays.

`free_array()`

Frees memory allocated to arrays.

`final():`

Take care of file clean up and complete the restart recovery process by calling `rettek_close()` and `free_array()`.

Scheduling Considerations

This program can be run weekly as well as ad hoc (for historic data). This program runs in phase 4.

The program needs to run after the on-order extract program for weekly runs.

Restart/Recovery

The logical unit of work (LUW) in this program is `dept/class/subclass`.

Store Add [storeadd]

Design Overview

This program will add all information necessary for a new store to function properly. When a store is added to the system, the store will be accessible in the system only after storeadd.pc is run.

The batch program loops through each record on the store_add table.

Also, it supports the replenishment system in RMS 9.0.

Scheduling Constraints

Processing Cycle: Daily, Ad Hoc Phase

Scheduling Diagram: N/A

Pre-Processing: pccxt.pc pcdnld.pc

Post-Processing: slocrbld.pc

Threading Scheme: Table based processing, don't use multithreading.

Restart/Recovery

Select ALL FIELDS from store_add.

After a record on store_add has been processed successfully, it is immediately deleted. Thus, restart recovery is implicit in storeadd.pc.

Program Flow

N/A

Function Level Description

init()

Declare restart variables

Get system variables (ELC indicator and pricing rule)

process()

Loop through store_add table

Set “new” variable indicators

Insert into store table

Call Insert_Pricing_Zone

If elc_ind = ‘Y’

 Call Insert_Cost_Zones

end if;

If repl_ind = ‘Y’

 Call Copy_Repl_info

end if;

If copy_close_ind = ‘Y’

 Call Copy_Close_Sched

End if;

If copy_dlvry_ind = ‘Y’

 Call Copy_Dlvry_Sched

End if;

Call Insert_Stock_Loc_Traits

Delete from store_add

Insert_Pricing_Zone()

This function inserts records into pricing zone tables as is appropriate to the store being created:

insert corporate pricing zone information

insert store pricing zone information

call Item_Zone_Price

if new_price_zone_ind = ‘N’

 insert zone info for existing currency

else

insert new zone info

call Item_Zone_Price (to add appropriate record for the new zone)

Insert_Cost_Zones()

This function inserts records into cost zone table as is appropriate to the store being created:

insert corporate cost zone information

insert store cost zone information

if new_cost_zone_ind = 'N'

insert cost zone detail records

else

insert new zone

Item_Zone_Price()

This function inserts records into the item_zone_price table for a new pricing zone after it's been created.

Copy_Store_Items()

This function calls the like_store_execute_sql.copy_store_items package function, which copies all item/store records from the like_store and inserts them for the new store.

Copy_Repl_Info()

This function copies all replenishment information for items from the selected like_store and copies them into replenishment tables for the new store.

Copy_Close_Sched()

This function copies all the location closed information from the selected like_store which the close_date are greater or equal to current and copies them into location_closed and company_closed_excep tables for the new store.

Copy_Dlvry_Sched()

This function copies all the location delivery schedules from the selected like_store and copies them into the loc_dlvry_sched, loc_dlvry_sched_days, and loc_dlvry_sched_exc tables for the new store.

Insert_Stock_Loc_Traits()

This function calls the `stkledgr_sql.stock_ledger_insert` and `loc_traits_sql.new_org_hier` package functions, which insert records into the stock ledger and hierarchy tables.

`final()`

This function stops restart recovery.

I/O Specification

N/A

Technical Issues

N/A

Supplier Constraint Scaling [supcnstr.pc]

Design Overview

This new batch program will process all orders eligible for scaling during the nightly replenishment run. This program will run nightly after the replenishment order build program (rplbld.pc) and before the replenishment purge program (rplprg.pc). The purpose of this program will be to select all of the orders created by the replenishment programs which are eligible for scaling. Once selected, the program will serve as a wrapper program and send each order number into the supplier constraint scaling library to actually perform the scaling on the order.

The orders which will be eligible for scaling are as follows:

If due order processing was used, only orders with a written date of today, origin type = 0 (replenishment order), due order processing indicator = 'Y', due order indicator = 'Y' and a scale order to constraint indicator = 'Y' will be processed. This encompasses all due orders created by replenishment which have constraints associated with them.

If due order processing was not used, only orders with a written date of today, origin type = 0 (replenishment order), status = 'A'pproved, due order processing indicator = 'N', due order indicator = 'Y', and a scale order to constraint indicator = 'Y' will be processed. This encompasses all approved orders created by replenishment which have constraints associated with them.

Scheduling Constraints

Processing Cycle: PHASE III (daily)

Scheduling Diagram: After rplbld and before rplprg. This is not a required program. It needs to be run if order scaling is required in the batch run. If scaling will only be done online, this program need not be run.

Pre-Processing: N/A

Post-Processing: None

Threading Scheme: SUPPLIER

Restart Recovery

The Logic Unit of Work for this program is an order number.

This program supports Retek standard intermittent commit based on a commit counter. The recommended commit counter is 1000. Once a commit counter number of records have been successfully processed, it will force a commit. If a given record fails, the error will be posted to the standard error log for the batch module. If the error is a non-fatal type, all subsequent records will continue to be processed. If a fatal error has occurred, all transactions from the last commit will be rolled back.

The driving cursor will need to be a union. The first section of the union will handle orders that are using due order processing. It should bring back every order where the `due_order_processing_ind` is 'Y' and the `scale_constr_ind` is 'Y'. The second section of the union will handle orders that are not using due order processing. It should bring back every order where the `due_order_processing_ind` is 'N' and the order is approved and the `scale_constr_ind` is 'Y'.

The program should be threaded by supplier.

Program Flow

TABLE	SELECT	INSERT	UPDATE	DELETE
ORDHEAD	Yes	No	Yes	No
ORD_INV_MGMT	Yes	No	Yes	No
PERIOD	Yes	No	No	No

Shared Modules

Order Scaling Library

Function Level Description

Main

Follows Retek standards for the main() function. Logs on to the database, calls init(), process(), and final().

Init

Calls retek_init() to start up restart recovery and bring back the bookmark string in case of a restart. Fetches vdate from the PERIOD table.

Process

This function is the controller for most of the processing in the program. It opens and fetches from the driving cursor according to Retek standards, using array processing and handling restart/recovery.

For each order returned by the driving cursor call the order scaling (scaling_cnstr) library which will perform the actual scaling of the order.

The scaling call will return success, fatal, and non-fatal errors. On success, the program will move on the next order. On fatal errors, the program will abort. Non-fatal errors will require extra processing. Handle_reject will be called for all orders with non-fatal errors that were in approved status. Update_oim will be called for all orders returning non-fatal errors from scaling.

Update_oim

A 2000 character error message will be populated by scaling for non-fatal errors. This error message will be placed in ORD_INV_MGMT.SCALE_ISSUES for the order being processed.

Handle_reject

If the order was in approved status, it will be updated to worksheet status and it reject reason will be updated to 'SO' (scaling issues). There orders orig_approval_id and orig_approval_date will also be set to NULL.

I/O Specification

None.

Supplier download [supdnld]

Design Overview

This program downloads supplier information to the warehouse. Triggers on the supplier and supplier address tables will fire when information has been added, modified, or deleted from the specified tables. When the trigger fires, it will write the supplier and action type to the staging table, lif_sup. This program will then read from the staging table and create a supplier file for the warehouse to process.

Address changes will be bridged for the primary address_type "03" only (RTV address).

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: This program does not have any scheduling constraints. It can be run multiple times in a batch schedule. The rms2rls.awk file conversion program should run after this program to properly interface with Nautilus.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

The Logical Unit of Work for the program will be a single row from the staging table, lif_sup. Restart/Recovery will be used for init and commit.

```
SELECT ls.action_type,
       to_char(ls.supplier),
       substr(s.sup_name, 1,28),
       substr(a.post, 1, 9),
       a.add_1,
       a.add_2,
       substr(a.add_3, 1, 12)||' '||substr(a.city, 1, 10)||' '||a.state||' '||a.country
```

```
_id,  
    TO_CHAR(s.qc_pct, '999d99'),  
    ls.rowid,  
    '||TO_CHAR(ls.supplier)  
FROM lif_supls ls,  
     sups s,  
     addr a  
WHERE s.supplier (+) = ls.supplier  
     and a.module (+) = 'SUPP'  
     and a.key_value_1 (+) = ls.supplier  
     and a.addr_type (+) = '03'  
     and a.primary_addr_ind (+) = 'Y'  
     and ls.supplier > NVL(:os_restart_supplier, -999)  
ORDER BY ls.supplier,  
         ls.rowid;
```

Program Flow

N/A

Function Level Description

The program will select the fields from the staging table and write the records to the flat file in the RLS format. Records that were successfully written to the flat file will be deleted from lif_sups, while records that failed to write will write a record to if_errors and will not be deleted from lif_sups.

Init()

Initialize restart recovery.

Open output file.

Write FHEAD record.

Process()

Open the driving cursor in a for loop.

Fetch the data consisting of all supplier information needed for the flat file.

Write the data to the FDETL line of the flat file

If the action_type = 'D'eleted, then only the supplier number and action type needs to be written to the file and sent down to Logistics.

Delete the record in lif_sups

final()

Write the FTAIL record.

Commit if no error, else rollback.

I/O Specification

Output File

The output file should be accepted as a runtime parameter at the command line.

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	Nine leading zeroes: 0000000001	ID of current line being processed by input file.
	File Type Definition	Char(4)	VNDD	Identifies file as 'Vendor download'
	File Create Date	Date	create date	date file was written by external system
Transaction Detail	File Type Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
	Action Type	Char(1)	lif_sups. action_type	'A'dd, 'M'odify, 'D'ele
	Location (DC)	Number(4)	NULL	not used
	Transaction Date/Time	Char (12) format: YYYYMMDDHHMI	sysdate	System date (YYYYMMDDHH24MI)
	Vendor Number	Char (7)	lif_sups.supplier	Supplier number
	Vendor Name	Char (28)	sups.sup_name	Supplier name
	Ship to Zip	Number (10)	sups_add.post_code	Zip code for primary RTV supplier.
	Vendor Shipping Address line 1	char (30)	sups_add.address_1	Address for primary RTV supplier
	Vendor Shipping Address line 2	char (30)	sups_add.address_2	Address for primary RTV supplier
	Vendor Shipping Address line 3	char (30)	sups_add.address_3 (11) sups_add.city (13) sups_add.state	City, state, and country for primary RTV supplier.

Record Name	Field Name	Field Type	Default Value	Description
			(3) supr_add.country_id (3)	
	Percent QA Sampling	Number (6) format: NNN.NN	supr.qc_pct	Quality control percent sampling. (eg. 025.00 is 25%)
File Trailer	File Type Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
	File Record Counter	Number(10)	determined internally	Number of records/transactions processed in current file (only records between head & tail)

Note that there is a space between fields in the Nautilus specified flat file format.

Note that RDM can only handle a vendor/supplier number that is no longer than 7 characters.

Note that both truncation and padding is required. In the case of padding, use the RMS field type to determining Left or Right padding (numeric: left, text right)

Error File

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

N/A

Sup_Data Amount Repository [supmth]

Design Overview

Accumulates SUP_DATA amounts by department/supplier/transaction type and creates or updates one SUP_MONTH row for each department/supplier combination. Based on the transaction type on SUP_DATA, the following fields on SUP_MONTH are updated:

- type 1 - purchases at cost (written for consignment sales and orders received at POS or online)
- type 2 - purchases at retail (written for consignment sales and orders received at POS or online)
- type 3 - claims at cost(written for claim dollars refunded on RTV orders)
- type 10 - markdowns at retail (net amount based on markdowns, markups, markdown cancellations and markup cancellations)
- type 20 - order cancellation costs (written for all supplier order cancellations)
- type 30 - sales at retail (written for consignment stock sales)
- type 40 - quantity failed (written for QC shipments with failed quantities)
- type 70 - markdowns at cost (net amount based on supplier cost markdowns)

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
SUP_DATA	No	Yes	No	No	No
SUP_MONTH	No	No	Yes	No	No

Indexes: none

Scheduling Constraints

Processing Cycle: PHASE 3

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: supmth_post(), records are truncated from the sup_data table.

Threading Scheme: DEPT

Restart Recovery

```

SELECT      sup_data.dept,
            TO_CHAR(sup_data.supplier),
            sup_data.tran_type,
            SUM(sup_data.amount),
            '||TO_CHAR(sup_data.dept)||
            '||TO_CHAR(sup_data.supplier)
FROM v_restart_dept rv,
sup_data
WHERE      rv.driver_value = sup_data.dept
AND      rv.driver_name = :ora_restart_driver_name
AND      rv.num_threads = :ora_restart_num_threads
AND      rv.thread_val  = :ora_restart_thread_val
AND      (sup_data.dept > NVL(:ora_restart_dept, -999) OR
(sup_data.dept = :ora_restart_dept AND
(sup_data.supplier > :ora_restart_supplier)))
GROUP BY   sup_data.dept,
            sup_data.supplier,
            sup_data.tran_type
ORDER BY   sup_data.dept,
            sup_data.supplier,
            sup_data.tran_type;

```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

N/A

I/O Specification

N/A

Technical Issues

N/A

Zone Security Rebuild [szonrbld]

Design Overview

The security features being added to RMS will be maintained in the batch cycle. With each run, the changes made to the data in RMS will be brought under the security features of RMS through the running of 3 batch programs. Szonrbld.pc will handle the maintenance for the zone security data. Zones will have different update/select attributes for a given user for any of a number of different functional areas like 'Pricing' or 'Clearances'. For each run, the program will use the security data defined for the user/group/functional area/pricing zone to define whether a user can select or update every zone covered by the defined rules. The functional document describes the architecture of the security features and how it works. If there are conflicting security definitions for a zone because a user is associated with more than one group, the user is, as expected, granted the capability. It will be multi-threaded by zone, and use restart_recovery.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
SEC_USER_GROUP	No	Yes	No	No	No
SEC_GROUP_ZONE_MATRIX	No	Yes	No	No	No
V_RESTART_ZONE_GROUP_ID	No	Yes	No	No	No
SYSTEM_VARIABLES	No	Yes	No	No	No
SEC_USER_ZONE_MATRIX	No	No	Yes	No	No

Scheduling Constraints

Processing Cycle: Daily

Scheduling Diagram: Must run batch program prepost.pc with parameters szonrbld pre , szonrbld.pc and prepost.pc with parameters szonrbld post in series.

Pre-Processing: Run prepost with parameters: szonrbld pre

Post-Processing: Run prepost with parameters: szonrbld post

Threading Scheme: Zone_group_id

Restart Recovery

The logical unit of work for zone security rebuild will be the user-functional area (column_code)-zone_group_id-zone_id. Restart/recovery will be based on the user-functional area (column_code)-zone_group_id-zone_id. The commit_max_ctr field should be set to prevent excessive rollback space usage.

Program Flow

N/A

Shared Modules

N/A

Driving Cursor:

```
SELECT u.user_id,
       z.column_code,
       z.zone_group_id,
       z.zone_id,
       MAX(z.select_ind),
       MAX(z.update_ind)
FROM sec_user_group u,
     sec_group_zone_matrix z,
     v_restart_zone_group_id v
WHERE u.group_id = z.group_id
     AND v.driver_value = z.zone_group_id
     AND v.num_threads = :pi_restart_num_threads
     AND v.thread_val = :pi_restart_thread_val
     AND (u.user_id > NVL(:ps_restart_user, '-999')
        OR (u.user_id = :ps_restart_user
            AND (z.column_code > :ps_restart_column_code
                OR (z.column_code = :ps_restart_column_code
```

```

AND (z.zone_group_id > :ps_restart_zone_group_id
OR (z.zone_group_id = :ps_restart_zone_group_id
AND (z.zone_id > :ps_restart_zone_id))))))
GROUP BY u.user_id, z.column_code, z.zone_id, z.zone_group_id;

```

Function Level Description

Main

Init()

Check SYSTEM_VARIABLES.update_loc_sec_ind. If the indicator is not set then the program exit normally without further processing.

Call retek_init() to get restart-recover variables.

Process()

Call size_rule_array() to allocate memory for arrays that store security rules.

Open the driving cursor in a while loop. Fetch the data into rule array.

Insert records in rule array into sec_user_zone_matrix.

Call restart_commit.

End of while loop.

Size_rule_array()

This function allocates memory for arrays that store security rules based on the maximum commit count set in table restart_control table. The rule array includes user_id, column_code, zone_group_id, zone_id, select_ind and update_ind.

Insert_array_to_table()

This function uses array insert to insert the contents in the rule array to table sec_user_zone_matrix.

final():

restart/recovery close

I/O Specification

N/A

Technical Issues

N/A

Size Ratio Build [szrtbld]

Design Overview

The szrtbld.pc program will read parameters setup on the size_ratio_head table for size_ratio_id's that were set up online and create size ratios for all size combinations that exist for the dept/class/subclass. These ratios will also be set up at the subclass/store level. When extracting sales history data, use sales history types (Clearance, Promotional, Regular) that have a value of 'Y' from the size_ratio_head table to calculate the ratios.

Only size_ratio_head records that are due to be rebuilt, or have been recently updated online (i.e. an update indicator set to 'Y') and are system generated (i.e. a system indicator set to 'Y') will be processed in the nightly run.

Size_ratio_detail records will be written for all locations for each size_ratio_head record that is rebuildable. These will hold sales per size / total sales for the subclass. There will also be size_ratio_detail records written for each store that the size_ratio's dept/class/subclass/sizes exist. These will hold sales per size per store / total sales for the subclass at the store.

When extracting sales history from the Retek history tables, if the beginning or end day to grab history lies in the middle of the week, then modify that date to be the previous end of week date.

For performance reasons, this program will write to an output file which can they be loaded into the Retek Tables when needed.

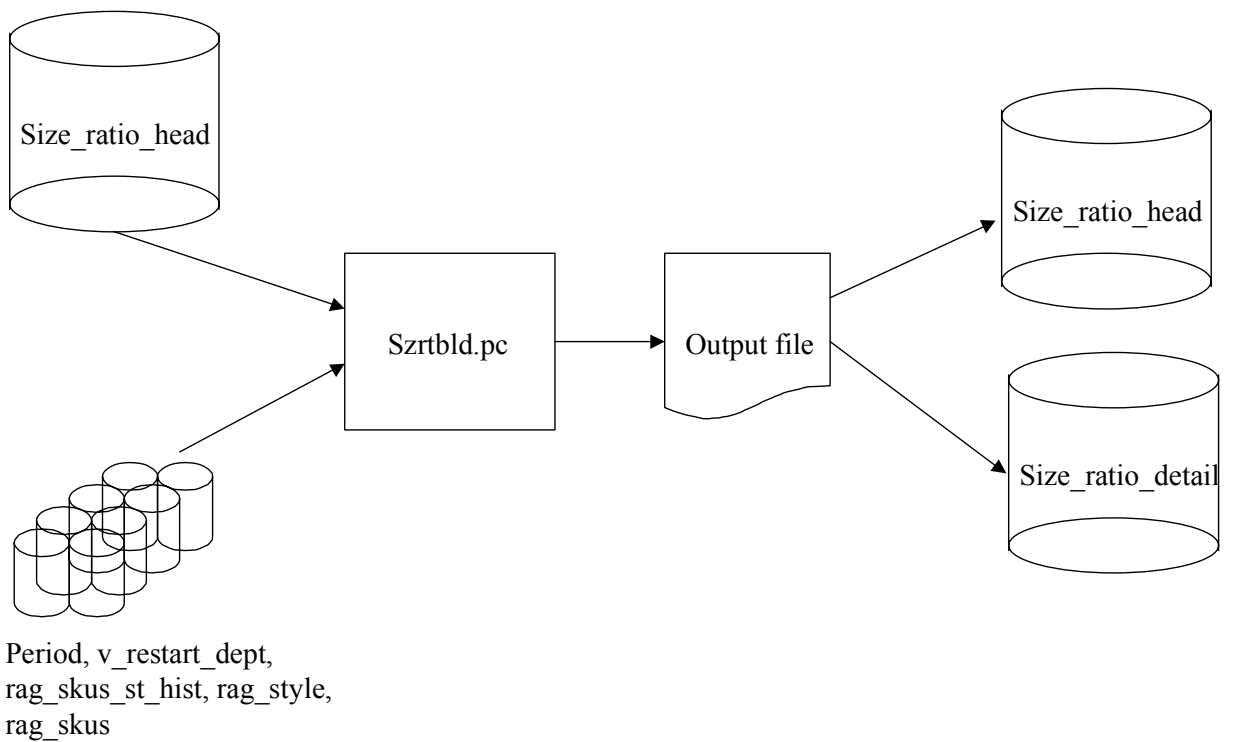
Scheduling Constraints

Processing Cycle:	PHASE 3 (daily)
Scheduling Diagram:	This program will likely be run after sales information is uploaded into Retek.
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

This program will be setup for multithreading and restart/recovery. The Logical Unit of Work will be at Department, Class, Subclass and will be threaded by department. Additionally, this program uses array processing to enhance performance. The arrays should be posted to the database every time the size field combination changes. The commit_max_ctr field on the restart_control table will determine the number of transactions that equal a logical unit of work.

Program Flow



Shared Modules

N/A

Function Level Description

Init()

This function will perform various initialization tasks as well as retrieve system parameters. The `restart_init()` function will initialize restart recovery for this program and populate program level restart recovery parameters. A call to the `size_arrays()` function will size the arrays used for SQL processing within the program. Linked lists used within the program are initialized to NULL and the output file is opened.

Process

This function will retrieve the parameters stored on `size_ratio_head` to build the `size_ratio_detail` records. Size Ratios will be built/rebuilt when the update indicator is Yes or the last review date plus the `review_weeks` is less than or equal to today. All functions needed to build `size_ratio_detail` records are called from this function and the restart/recovery and commit logic is performed here.

Update_szrt_head

This function will reset the update indicator to No and the last review date to today for the current `size_ratio_head` record.

Delete_szrt_detail

This function deletes all `size_ratio_detail` records for the current size ratio id. This program always deletes the `size_ratio_detail` records and reinserts new records. This streamlines the program and easily allows the program to account for new sizes that may have been added since the size ratios were last built for the size ratio id.

Build_store_list

This function extracts the sales history for the subclass that is determined by the `size_ratio_head` parameters and feeds them into a linked list.

Set_cursor_options

This function prepares the `size_ratio_head` parameter variables for use in the extract of sales history.

Free_store_list_memory

This function releases memory allocated to the store linked list.

Add_store_to_list

This function will add stores to the store linked list for processing.

Process_store_list

This function will write the information to the SQL loader file.

Compare_sizes

This function will take in a size group/size combination and determine whether the combination already exists in the linked list, or if it should be added.

Build_subclass_list

This function will take the size group/size combinations and create an ordered link list for faster processing.

Records are added in their appropriate spot in the list. The total sales for the subclass is also incremented.

Process_subclass_list

This function calculates the subclass level size ratios and feeds them into the SQL loader file.

Free_subclass_list_memory

This function releases memory used in the subclass linked list.

Size_arrays

This function will initially size the arrays used for SQL fetches, inserts, and updates. The default size for all arrays in this program is the restart max counter on the restart control table.

Final

This function will complete restart/recovery processing, close the output file, and erase the temporary output file.

I/O Specification

Output File

The output file is a SQL loader file that can be loaded into Retek at anytime.

Field Name	Field Type	Description
Size Ratio ID	Num(6)	Contains the ID for the Size Ratio
Sequence Number	Num(4)	Used to form the Primary Key with the Size Ratio ID
Store	Num(4)	The ID for the store
Size Group 1	Num(4)	The Size Group for which the first size belongs to
Size 1	Char(6)	The ID for the first size
Size Group 2	Num(4)	The Size Group for which the second size belongs to
Size 2	Char(6)	The ID for the second size
Sales	Num(12,4)	Historic sales associated with the size combination
Size Ratio	Num(12,4)	The percentage of sales this size combination contributes to sales

Technical Issues

This program writes to a SQL Loader file, instead of a table for performance reasons. Because of the size of the rag_skus_st_hist, and the amount of database hits against it, it this program allows the flexibility of loading the Size Ratio Detail records into the RMS when appropriate.

SKU/Location Ticket Output File [tcktdnld]

Design Overview

This program will create an output file containing all of the information to be printed on a ticket or label for a particular SKU/location. This program is driven by the “requests” for tickets that exist on the TICKET_REQUEST table. Information to be printed on the ticket is then retrieved based on the SKU, location and the ticket type requested. The details, which should be printed on each type of ticket, are kept on the TICKET_TYPE_DETAIL table. Specific details, which will be written to the output file, are taken from the various item tables (i.e. SKU short description from WIN_SKUS, retail price from SKU_ZONE_PRICE).

Scheduling Constraints

Processing Cycle:	Ad Hoc (Daily)
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

Restartability will exist implicitly within this program. Because records will be deleted after they are selected, no explicit code is needed to restart in the event of a failure.

The lack volume of data processed by this program, in addition to the lack of an appropriate threading mechanism, negates the need for Retek multi-threading capabilities.

Driving Cursor:

```
SELECT tr.ticket_type_id,
       tr.sku,
       tr.quantity,
       tr.loc_type,
       tr.location,
       tr.country_of_origin,
       tr.unit_retail,
       tr.multi_units,
       tr.multi_unit_retail,
       ROWIDTOCHAR(tr.rowid)
FROM ticket_request tr
ORDER BY 1, 2, 5;
```

Program Flow

N/A

Shared Modules

N/A

Function Level Description

init() –

Functional details:

This function should initialize the restart/recovery process. The output file should be opened, and if it is not a “restart”, then file header information should be written. The system vdate is selected for the file create date used in the output file header. The format_buffer function should be called to format output strings. The size_arrays function should be called to size the fetch and delete arrays.

Technical details:

The output file should be written in the style necessary for Retek restart/recovery. That is, a temp file should be opened and initialized, and the final file should only be written to when the restart/recovery commit logic is called (using `restart_file_write`).

`process()` –

Functional Details:

This function should write transaction records to the output file for each sku/ticket type/location combination on the `ticket_request` table. For each record a transaction header should be written to the output file and the `ticket_item` function should be called to write the detail items for the details associated with the ticket type. If the item is a pack item, however, the `ticket_pack` function should be called to first write component sku records (the `ticket_item` function will be called within the `ticket_pack` function for each component sku.). After each record from the driving cursor is processed, it will be deleted from the `ticket_request` table. Finally, when all of the records have been processed from the table, a file trailer should be written to the output file.

Technical Details:

The function should fetch records from the driving cursor into arrays. The arrays should be sized to match the value of the maximum commit counter on the `restart_control` table. Once the records are fetched, each record should be processed in a for-loop. After all of the records have been processed in the for loop, the records should be array deleted from the `ticket_request` table by rowid, and the `restart_commit` logic should be called. Output file line counters, transaction counters, etc. should be saved into the application image array string that is passed to the `restart_control` function.

`ticket_pack()`

This function will be called from `process` if the sku on the `ticket_request` table is a pack item. This function should fetch all of the component skus in the pack, along with pack quantity information, and write a pack record for each component. Further the `ticket_item` function should be called for each component.

`ticket_item()`

This function should select all of the records from the `ticket_type_detail` table with the `ticket_type` from the `ticket_request` record. Detail records should be written out to the output file for each detail record retrieved. Either item information or attribute information should be written to the output file. If the ticket item is to be written (fetched ticket item is not null) `get_ticket_item` is called to retrieve this appropriate information. If the attribute information is to be written (fetched attribute column is not null) then a function should be called to get the appropriate attribute information (`get_UDA`).

`get_ticket_item()`–

This function retrieves the database information which corresponds to the requested ticket item, according to the table below.

TICKET ITEM	OUTPUT FILE VALUE
UOM	Price per unit of measure from win_skus or rag_skus.
SKU	Retek SKU value
SKUD	SKU description (from desc_look)
SKSD	SKU short description from win_skus table
UPC	UPC & supplement from upc_ean table
SIZ1	Size1 value from rag_skus
SIZ2	Size2 value from rag_skus
COLR	Color value from rag_skus
WGHT	Carton weight from item/style table
DEPT	Department from desc_look & department name from deps table
CLAS	Class value from item/style table & class name from class table
SBCL	Subclass from item/style table & subclass name from subclass table
RTPC	Retail price from driving cursor (if available), otherwise from sku_zone_price for sku/store (use base zone value for warehouses).
SRTP	Suggested retail price from rag_style
MUPC	Multi-units and multi-unit retail from driving cursor (if available), otherwise from sku_zone_price for sku/store (use base zone value for warehouses)
SUPR	Supplier from ordhead for most recent PO for the SKU.
SUPC	Supplier color, from rag_skus
SUPS	Supplier size, from rag_skus.
STRE	Store from driving cursor
WHSE	Warehouse from driving cursor
COOG	Country of origin from driving cursor if available, else from the last PO (see supplier).

get_UDA()

This function should fetch the user defined attribute (UDA) value and description associated with the attribute value selected from the ticket detail table. The UDA description will be selected for the UDA and the SKU from either the UDA_item_lov and the UDA_value tables, the UDA_item_ff table or the UDA_item_date table. The UDA value will be written to the output file in the “value” location of the detail line.

final()

Retek restart/recovery process will be closed by calling the internal API function, and all appropriate output files will be close and temp files will be removed.

I/O Specification

Output File:

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence	Number(10)		Line number of the current file
	File Type Definition	Char(4)	TCKT	Identifies file as ‘Print Ticket Requests’
	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 Sequence	Number(10)		Line number of the current file
	SKU	Number(8)		ID number of the staple or fashion SKU or the pack item
	Ticket Type	Char(4)		ID which indicates the ticket type to be printed

Record Name	Field Name	Field Type	Default Value	Description
	Location Type	Char(1)	S - Store W - Warehouse	Identifies the type of location for which tickets will be printed
	Location	Number(4)		number of the store or warehouse for which tickets will be printed
	Quantity	Number(12,4)		the quantity of tickets to be printed
Transaction Component	File Type Record Descriptor	Char(5)	TCOMP	Identifies file record type
	File Line Sequence	Number(10)		Line number of the current file
	SKU	Number(8)		ID number of the SKU
	Quantity	Number(12,4)		quantity of the component SKU as part of the whole; if SKU on the header record is a staple or fashion SKU, the value in this field will be 1.
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence	Number(10)		Line number of the current file
	Detail Sequence Number	Number(10)		Sequential number assigned to the detail records
	Ticket Item	Char(4)		ID indicating the detail to be printed on the ticket
	Attribute Description	Char(40)		Description of the attribute (from the UDA Table)

Record Name	Field Name	Field Type	Default Value	Description
	Value	Char(100)		Detail to be printed on the ticket (i.e. UPC, Department Number, SKU description)
	Supplement	Char(300)		Supplemental description to the Value (i.e. Department Name)
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Sequence	Number(10)		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 Sequence	Number(10)		Line number of the current file

Technical Issues

The program could be sped up by outer joining ticket_type_detail into the driving cursor, and avoiding the c_ttd cursor which must be opened for each record in our fetch array.

[tifposdn.pc]

Design Overview

This batch program will process records off of the TIF_EXPLODE table, calculate a tax rate total for every item/department/site combination and write this total to a POS flat file.

Scheduling Constraints

Processing Cycle:	Phase 4 (daily)
Scheduling Diagram:	N/A
Pre-Processing:	txrposdn.pc
Post-Processing:	prepost.pc (to TRUNCATE tif_explode table)
Threading Scheme:	N/A

Restart Recovery

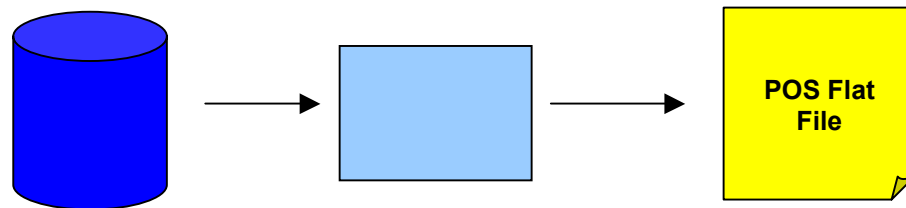
Logical Unit of Work = item/department/store combination

Driving Cursor:

Change this cursor to :

```
EXEC SQL DECLARE c_tif_explode CURSOR FOR
    SELECT store,
           sku,
           dept,
           current_rate_ind,
           changed_ind
    FROM tif_explode
    ORDER BY store,sku;
```

Program Flow



Shared Modules

None

Function Level Description

Init()

Call retek_init to initialize file processing and restart/recovery.

Get the system date

Size all arrays needed for data processing.

Create format strings for writing FHEAD, FDETL, and FTAIL lines.

Process()

Functional Details:

This function drives the processing of the program. Data will be read from the tif_explode table and each record on the tif_explode table will be processed into a transaction record on the output file. This file will contain separate transactions for every store listed on the tif_explode table.

Fetch data from the driving cursor.

For every new store, write a new THEAD line (new transaction) If necessary, write a TTAIL line for the record block written for the previous store.

Calculate the tax rate for the item by calling the calc_tax function. (See the description of the calc_tax for details on how the tax rate will be calculated based on the values of changed_ind and current_rate_ind.

When all records have been read from tif_explode, write the FTAIL line.

Final()

Close restart/recovery with a call to retek_close.

Free memory previously allocated to arrays.

Tax Rate Calculation

Data structures needed for tax rate calculation:

```
struct geocode_buffer{

    char (rs_geocode_level)[LEN_GEOCODE_LEVEL],

        (rs_country_geocode_id)[LEN_GEOCODE_ID],

        (rs_state_geocode_id)[LEN_GEOCODE_ID],

        (rs_county_geocode_id)[LEN_GEOCODE_ID],

        (rs_city_geocode_id)[LEN_GEOCODE_ID],

        (rs_district_geocode_id)[LEN_GEOCODE_ID];

}geocode_data;
```

This structure is used to organize geocode data for a particular store.

```
struct tax_data_buffer{

    char (*as_tax_jurisdiction_id)[LEN_TAX_JURIS],

        (*as_tax_type_id)[LEN_TAX_TYPE];

}tax_data;
```

Cursors used for tax rate calculation.

Before processing, we need to fetch the geocode_id and geocode_level from the geocode_store table. The cursor c_get_geocode_store will fetch geocode hierarchy information from the geocode_store

```
EXEC SQL DECLARE c_get_geocode_store CURSOR FOR

SELECT geocode_level,

        country_geocode_id,

        state_geocode_id,
```

```
        county_geocode_id,  
        city_geocode_id,  
        district_geocode_id  
FROM geocode_store  
WHERE store = :is_store;
```

A single cursor (c_get_txcode) is defined to fetch tax rates that

Match the tax jurisdiction and type IDs for the given item.

Apply to the item across multiple (CNTRY, STATE, etc.) levels.

```
EXEC SQL DECLARE c_get_txcode_item CURSOR FOR
```

```
    SELECT ptc.tax_jurisdiction_id,  
           ptc.tax_type_id
```

```
    FROM product_tax_code ptc
```

```
    WHERE ptc.item = :is_sku
```

```
           AND ptc.item is NOT NULL
```

```
INTERSECT
```

```
    SELECT gt.tax_jurisdiction_id,  
           gt.tax_type_id
```

```
    FROM geocode_txcode gt
```

```
    WHERE  gt.country_geocode_id = :geocode_data.rs_country_geocode_id
```

```
           AND ( gt.state_geocode_id = :geocode_data.rs_state_geocode_id
```

```
                 OR gt.state_geocode_id IS NULL)
```

```
           AND ( gt.county_geocode_id = :geocode_data.rs_county_geocode_id
```

```
                 OR gt.county_geocode_id IS NULL)
```

```
           AND ( gt.city_geocode_id = :geocode_data.rs_city_geocode_id
```

```
                 OR gt.city_geocode_id IS NULL)
```

```

AND ( gt.district_geocode_id = :geocode_data.rs_district_geocode_id
      OR gt.district_geocode_id IS NULL);

```

The cursor `c_get_txcode` will get the tax code that applies to the item/dept in question by retrieving data that applies to

The tax code for the specific item in question (`product_tax_code`)

Any taxes that may apply to the item on the basis of geocode (i.e. location)

```
EXEC SQL DECLARE c_get_txcode CURSOR FOR
```

```
SELECT ptc.tax_jurisdiction_id,
```

```
       ptc.tax_type_id
```

```
FROM product_tax_code ptc
```

```
WHERE (ptc.item = :is_sku
```

```
       AND dept IS NULL)
```

```
OR (ptc.dept = :is_dept
```

```
    AND item IS NULL)
```

```
INTERSECT
```

```
SELECT gt.tax_jurisdiction_id,
```

```
       gt.tax_type_id
```

```
FROM geocode_txcode gt
```

```
WHERE  gt.country_geocode_id = :lt_geocode_data.rs_country_geocode_id
```

```
AND ( gt.state_geocode_id = :lt_geocode_data.rs_state_geocode_id
```

```
     OR gt.state_geocode_id IS NULL)
```

```
AND ( gt.county_geocode_id = :lt_geocode_data.rs_county_geocode_id
```

```
     OR gt.county_geocode_id IS NULL)
```

```
AND ( gt.city_geocode_id = :lt_geocode_data.rs_city_geocode_id
```

```
     OR gt.city_geocode_id IS NULL)
```

```
AND ( gt.district_geocode_id = :lt_geocode_data.rs_district_geocode_id
```

OR gt.district_geocode_id IS NULL);

Declare C_GET_TAX_RATE that selects the tax_rate from the tax_rate table where:

tax_jurisdiction_id and tax_type_id are equal to the tax_jurisdiction_id and tax_type_id that were fetched from the geocode cursor

AND the start date of the tax_rate is less than, or equal to, L_date and the end_date is greater than, or equal to, the date for which the tax is being calculated.

EXEC SQL DECLARE c_get_tax_rate CURSOR FOR

SELECT tax_rate

FROM tax_rates

WHERE tax_jurisdiction_id = :tax_data.tax_jurisdiction_id[ll_ctr]

AND tax_type_id = :tax_data.tax_type_id[ll_ctr]

AND (TO_DATE):is_date, 'YYYYMMDD')

BETWEEN start_date

AND end_date);

int calc_tax(long *iol_tax_rate,

char *is_date,

char *is_sku,

short ii_null_sku,

char *is_dept,

short ii_null_dept,

char *is_store)

Loop through records on TIF_EXPLODE

If the current_ind = 'Y' and the changed_ind = 'N' then set L_date = current date. This will cause the current tax rate for the item/store combination to be sent to POS.

If the current_ind = 'Y' and the changed_ind = 'Y' then set L_date = current date. DO NOT loop through any tax cursors or retrieve any tax rates. In this case, a tax rate of 0% for the current date needs to be written to the POS file.

If the current_ind = 'N' and the changed_ind = 'N' then set L_date = (current date + 2). This will send the rate that will take effect in two days to POS.

If the current_ind = 'N' and the changed_ind = 'Y' loop through the C_GET_TAX_RATE cursor for L_date = current date and L_date = (current date + 1). This will send down two tax rate records to POS. One will be for today and one for tomorrow. This is to cover the case that a tax rate or code is modified after the batch process has already sent the record to POS.

Retrieve the geocode hierarchy for the store from the geocode_store table

If the store is not found on the geocode_store table then write the record to the error log file.

Based on the geocode level retrieved, loop through the appropriate Tax Code cursor to get the geocode tax codes geocode_txcode.tax_jurisdiction_id = product_tax_code.jurisdiction_id and geocode_txcode.tax_type_id = product_tax_code.tax_type_id for that item. (see above cursors)

After each tax code combination is retrieved fetch the tax rate from the tax_rates table where tax_rates.tax_jurisdiction_id = tax_jurisdiction_id fetched from the geocode tax code cursor and tax_rates.tax_type_id = tax_type_id fetched from the geocode tax code cursor. (see above cursors)

Calculate the total tax rate (L_total_tax_rate := L_total_tax_rate + C_rec.tax_rate) while looping through taxcodes and write item/deptartment/store/tax rate sum to the POS flat file.

I/O Specification

Output file: Format will be as in table given below.

All character variables should be right-padded with blanks and left justified; all numerical variables should be left-padded with zeroes and right-justified.

Missing variables should be blank.

Record	Variable	Field Type	Default Value	Description
FHEAD	File record descriptor	CHAR(5)	FHEAD	Identifies file record type.
	Line number	NUMBER(10)	0000000001	Identifies the line number
	Program descriptor	CHAR(8)	tifposdn	Identifies the program.
	Create date	CHAR(14)	File create date	YYYYMMDDH H24MISS format
THEAD	File record descriptor	CHAR(5)	THEAD	Identifies Transaction Header
	Line Number	NUMBER(10)		Sequential Line Number
	Transaction Number	NUMBER(10)		Transaction Number
	Store	Numeric ID (4)		Identifies store number
TDETL	File Record descriptor	CHAR(5)	TDETL	Identifies transaction detail
	Line Number	NUMBER(10)		Sequential Line Number
	Transaction Number	NUMBER(10)		Transaction Number
	Department	NUMBER(4)		Department number.
	SKU	NUMBER(8)		Identifies the item.
	Tax Rate	NUMBER(8)		A new sales tax associated with the item (% = /5)

Record	Variable	Field Type	Default Value	Description
	Start Date	CHAR(8)		Start date for the new tax rate. (in YYYYMMDD format)
TTAIL	File Record Descriptor	CHAR(5)	TTAIL	Identifies transaction trailer.
	Line Number	NUMBER(10)		Sequential line number
	Transaction Number	NUMBER(10)		Transaction Number
	Detail Counter	NUMBER(6)		Number of detail records in transaction.
FTAIL	Line Number	NUMBER(10)		Sequential Line Number
	Detail Line Counter	NUMBER(6)		Total Number of detail records processed.

Transportation Upload [tranupld.pc]

Design Overview

The purpose of the TRANUPLD.pc batch program is to update the TRANSPORTATION table. This will allow the users to view and implement the transportation data online instead of manually viewing and inserting information.

This upload process will diverge from the current RMS 8.0 upload programs in that it will write both fatal and non fatal errors to the IF_ERRORS table rather than a reject file. The IF_ERRORS table holds the program that caused the error (trnupld.pc in this case), error date (vdate), unit of work (Vessel ID, Voyage ID, Estimated Departure Date, Order No., Item, Container ID) and the error text.

This upload program has two business rules that need to be enforced. First, a PO/Item combination will be associated with a single Invoice. Second, a Container will be associated with a single Bill of Lading. These business rules will allow batch driven updates of the transportation table to function more smoothly and in a more time efficient manner.

Upload Files

Four record types will be used in this program's upload files: FTRAN, DTRAN, DPOIT, and FTAIL. The FTRAN record will contain general header information. All pieces of the unique key, other than PO/Item combinations, will be found within DTRAN records. The remaining DTRAN information represents the details pertaining to the current vessel. Each DTRAN record must be followed by at least one DPOIT record. DPOIT records will store the remaining portion of the key (mainly, the PO/Item combination). Finally, the file will end with an FTAIL record that has a field holding the total number of records in the file.

For the purpose of this design a unit of data will be defined as the collection all the information in a given DPOIT record as well as the data in that of the DTRAN record to which it is associated. Depending on the `acd_code` found in each DPOIT record, these units of data will be either added, deleted, or used to update a record in the transportation table.

Program Flow

Read current record into appropriate data structure

This format process should include checks which verify that all date fields and all numeric fields contain data of the appropriate type.

Validate current record

Certain fields must contain values that already appear in other locations in the database. Ensure that these fields contain valid data.

Process current record

Processing will only occur when the current record is of type DPOIT. Use the `acd_code` to determine what action to perform on the transportation table.

Tables Used

TABLE	SELECT	INSERT	UPDATE	DELETE
TRANSPORTATION	Yes	Yes	Yes	Yes
IF_ERRORS	No	Yes	No	Yes
PERIOD	Yes	No	No	No

Scheduling Constraints

Pre/Post Logic Description

Processing Cycle: PHASE 4 (may also be schedule ad hoc to run multiple times per day)

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

Logical Unit of Work (recommended Commit checkpoints)

A valid DTRAN record.

Shared Modules

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

TRANSPORTATION_SQL.GET_NEXT_ID

This function will create a new Transportation ID for the TRANSPORTATION table.

Function Level Description

All database interactions required and error handling considerations

init():

Open input file.

Get vdate.

Initialize restart/recovery.

Get FTRAN and validate.

file_process():

Format a record from the input file.

Determine if the record is valid by calling the validate_record() function.

If the record is a valid DPOIT, call the process_record() function to perform an insert, update or delete statement.

Continue processing until an FTAIL record is found

format_record():

If the record is a 'DTRAN' record call the format_dtran_record() function.

If the record is a 'DPOIT' record call the format_dpoit_record() function.

If the record is a 'FTAIL' record call the format_ftail_record() function.

format_dtran_record():

Read dtran from input file. Update indicator values for null fields. Ensure that consolidator and candidate ind fields are not null. If the candidate field is null, populate it with an 'N'. Validate that all date fields and all numeric fields contain data of the appropriate type.

format_dpoit_record():

Read dpoit from input file. Update indicator values for null fields. Ensure that the rush ind, order and item are not null. If the rush ind is null, populate it with an 'N'. Validate that all date fields and all numeric fields contain data of the appropriate type.

format_ftail_record():

Read ftail from input file. Ensure that the actual number of records processed matched the stated number of records.

validate_record():

If the record is a 'DTRAN' record call the validate_dtran_record() function.

If the record is a 'DPOIT' record call the validate_dpoit_record() function.

If the record is a 'FTAIL' record call the validate_ftail_record() function.

validate_tran_record():

vessel_id, voyage_flt_id and estimated_depart_date must be all have a value, or must all be null. If this condition is not met, reject the record.

validate the consolidator by calling a function which will return true when the given consolidator = partner_id on the partner table with partner_type = 'CO'.

validate the trans mode by calling a function which will return true when the given trans mode = code_detail on the code_detail table with code_type = 'TRMO'.

validate the vessel_scac_code by calling a function which will return true when the given vessel_scac_code = scac_code on the scac table

validate the lading port by calling a function which will return true when the given lading port = outloc_id on the outloc table with outloc_type = 'LP'.

validate the discharge port by calling a function which will return true when the given discharge port = outloc_id on the outloc table with outloc_type = 'LP'.

validate that the estimated departure date is <= actual arrival date <= delivery date.

validate the container_scac_code by calling a function which will return true when the given container_scac_code = scac_code on the scac table

validate the freight type by calling a function which will return true when the given freight type = freight_type on the freight_type table

validate the freight size by calling a function which will return true when the given freight size = freight_size on the freight_size table

validate_poit_record():

validate the order number by calling a function which will return true when the given order_no = order_no on the ordhead table

validate the item by calling a function which will return true when the given item = item on the desc_look table

validate the order/item combination by first checking if the combination exists, as is, on the ordsku table. If it does not, then check that the given item is a style for any item associated with the given order.

validate the origin_country_id by calling a function which will return true when the origin_country_id = country_id on the country table

validate the consolidation_country_id by calling a function which will return true when the consolidation_country_id = country_id on the country table

validate the export_country_id by calling a function which will return true when the export_country_id = country_id on the country table

validate the status by calling a function which will return true when the given status = code_type on the code_detail table with code_desc = 'TRCO'.

validate the packing type by calling a function which will return true when the given packing type = code_type on the code_detail table with code_desc = 'PKMT'.

validate the item quantity by calling a function which will return true when the given item quantity = uom on the uom_class table with uom_class = 'QTY'.

validate the carton quantity by calling a function which will return true when the given carton quantity = uom on the uom_class table with uom_class = 'PACK'.

validate the gross weight by calling a function which will return true when the given gross weight = uom on the uom_class table with uom_class = 'MASS'.

validate the net weight by calling a function which will return true when the given net weight = uom on the uom_class table with uom_class = 'MASS'.

validate the cubic uom by calling a function which will return true when the given cubic uom = uom on the uom_class table with uom_class = 'VOL'.

validate the currency code by calling a function which will return true when the currency code = currencies on the currencies table.

Check that the current record type is valid given the preceding record type. FTRAN records can only occur at the beginning of the file. DPOIT records may only follow DPOIT or valid DTRAN records. DTRAN records can only follow FTRAN or DPOIT records. If a DTRAN valid is rejected, all DPOIT records associated with it must also be rejected.

Validate that if a carton quantity exists, then a carton quantity uom also exists. If a cubic exists, then a cubic uom exists. If an invoice amount exists, then a currency code exists. If a gross weight exists, then a gross weight uom exists. If a net weight exists, then a net weight uom exists. If any of these conditions are not satisfied then reject the record.

process_record():

Since the TRANSPORTATION table is completely de-normalized we need to synthesize the multiple records on the input files into single records to insert or update into the table or find the proper where clause to delete from it. To determine whether or not a combination exists on the TRANSPORTATION table call the appropriate cursor (there are four of them, one with PO/Item, another with VVE,PO/Item, another Container,PO/Item and the last VVE,Container,PO/Item).

Validation must prevent the following unique key from being violated:

vessel_id / voyage_flt_id / estimated_depart_date,

order_no / item,

container_id

If any value on one of the above lines is null, then all values on that line must also be null.

if the DPOIT record does exist then

if the acd_code = 'A' then

validate that if candidate indicator is “Y” then invoice id, invoice amt, invoice date, currency code, exchange rate, item quantity, and item quantity uom are not null.

to determine if the unique key combination including PO/Item already exists in the TRANSPORTATION table.

if the combination does not exist on the TRANSPORTATION table then

build an insert statement with DTRAN and DPOIT data

if the combination does already exist on the TRANSPORTATION table then

write error

continue on to the next DPOIT, DTRAN or FTAIL section of the file

if the acd_code = ‘C’hange then

validate that if candidate indicator is “Y” then invoice id, invoice amt, invoice date, currency code, exchange rate, item quantity, and item quantity uom are not null.

if the DTRAN Vessel_id, Voyage_flt_id, Estimated_depart_date and Container_id fields are NULL then

that will validate if the PO/Item combination already exists on the TRANSPORTATION table

if the PO/Item combination already exists on the TRANSPORTATION table then

build an update statement with DTRAN and DPOIT data

if the PO/Item combination with all others NULL does not exist on the TRANSPORTATION table then

write error

continue on to the next DPOIT, DTRAN or FTAIL section of the file

if the DTRAN Vessel_id, Voyage_flt_id, Estimated_depart_date fields are NULL and the Container_id field is not NULL then

that will validate if the PO/Item, Container combination already exists on the TRANSPORTATION table with VVE NULL

if the PO/Item, Container combination already exists on the
TRANSPORTATION table then

build an update statement with DTRAN and DPOIT data

if the PO/Item, Container combination does not exist on the
TRANSPORTATION table then

that will validate if the PO/Item combination already exists on the
TRANSPORTATION table with all others NULL

if the PO/Item combination already exists on the TRANSPORTATION table
then

build an update statement with DTRAN and DPOIT data

if the PO/Item combination does not exist on the TRANSPORTATION table
then

write error

continue on to the next DPOIT, DTRAN or FTAIL section of the file

if the DTRAN Container_id field is NULL and Vessel_id, Voyage_flt_id,
Estimated_depart_date fields are not NULL then

that will validate if the PO/Item, VVE combination with Container NULL,
already exists on the TRANSPORTATION table

if the PO/Item, VVE combination already exists on the TRANSPORTATION
table then

build an update statement with DTRAN and DPOIT data

if the PO/Item, VVE combination does not exist on the TRANSPORTATION
table then

that will validate if the PO/Item with all others NULL, combination already
exists on the TRANSPORTATION table

if the PO/Item combination already exists on the TRANSPORTATION table
then

build an update statement with DTRAN and DPOIT data

if the PO/Item combination does not exist on the TRANSPORTATION table
then

write error

continue on to the next DPOIT, DTRAN or FTAIL section of the file

if the DTRAN Vessel_id, Voyage_flt_id, Estimated_depart_date fields are not NULL and the Container_id field is not NULL then

that will validate if the PO/Item, VVE, Container combination already exists on the TRANSPORTATION table

if the PO/Item, VVE, Container combination already exists on the TRANSPORTATION table then

build an update statement with DTRAN and DPOIT data

if the PO/Item, VVE, Container combination does not exist on the TRANSPORTATION table then

that will validate if the PO/Item, Container, VVE NULL, combination already exists on the TRANSPORTATION table

if the PO/Item, Container combination already exists on the TRANSPORTATION table then

build an update statement with DTRAN and DPOIT data

if the PO/Item, Container, VVE NULL combination does not exist on the TRANSPORTATION table then

that will validate if the PO/Item, VVE, Container NULL combination already exists on the TRANSPORTATION table

if the PO/Item, VVE, Container NULL combination already exists on the TRANSPORTATION table then

build an update statement with DTRAN and DPOIT data

if the PO/Item, VVE, Container NULL combination does not exist on the TRANSPORTATION table then

that will validate if the PO/Item, VVE NULL, Container NULL combination already exists on the TRANSPORTATION table

if the PO/Item, VVE NULL, Container NULL combination already exists on the TRANSPORTATION table then

build an update statement with DTRAN and DPOIT data

if the PO/Item, VVE NULL, Container NULL combination does not exist on the TRANSPORTATION table then

write error

continue on to the next DPOIT, DTRAN or FTAIL section of the file

if the acd_code = 'D'elele then

to see if the Vessel/Voyage/ETD/PO/Item/Container/BL/Invoice combination exists on the TRANSPORTATION table

if the Vessel/Voyage/ETD/PO/Item/Container/BL/Invoice combination exists on the TRANSPORTATION table.

build a delete statement with DTRAN and DPOIT data

if the combination does not exist on the TRANSPORTATION table then

write error

continue on to next DTRAN or FTAIL section of the file

I/O Specification

All files layouts input and output

The following file formats will be used:

Header - DTRAN - Vessel/Voyage/ETD/Container/BL/Invoice

Detail - PO/Item

File Format – Vessel/Voyage/ETD/Container/BL/Invoice

Key

Italicized field names must be included in the input file.

Bold field names are part of the primary key

Record Name	Field Name	Field Type	Default Value	Description
FTRAN	Record descriptor	Char(5)	FTRAN	File head marker
	Line id	Char(10)	0000000001	Unique line id
	File type definition	Char(8)	TRANUPLD	Identifies program to use
	File create date	Char(8)	Current date	YYYYMMDD format
DTRAN	Record descriptor	Char(5)	DTRAN	Vessel, Voyage, ETD, Container, BL, Invoice File head
	Line id	Char(10)		Unique line id
	<i>Consolidator</i>	<i>Char(10)</i>		<i>Identifies the Consolidator. Validated against PARTNER table with type = 'CO'</i>
		Char(20)		Identifies the Vessel
	Voyage ID	Char(10)		Identifies the Voyage or Flight ID
	Estimated Depart Date	Char(8)		YYYYMMDD format
	Shipment Number	Number(10)		Identifies an outside Shipment number
	Actual Arrival Date	Char(8)		YYYYMMDD format
	Trans Mode	Char(6)		Identifies the type of transportation being used. Valid values are found in the TRMO Code Type on the CODE DETAIL table
	Vessel SCAC Code	Char(6)		Customs defined ID for the Vessel. Validated against SCAC table.
	Estimated Arrival Date	Char(8)		YYYYMMDD format
	Lading Port	Char(5)		Identifies the Lading Port. Validated against OUTLOC with type = 'LP'

Record Name	Field Name	Field Type	Default Value	Description
	Discharge Port	Char(5)		Identifies the Discharge Port. Validated against OUTLOC with type = 'DP'
	Service Contract Number	Char(15)		Identifies the outside Service Contract Number
		Char(20)		Identifies the Container
	Container SCAC code	Char(6)		Customs defined id for the container. Validated against SCAC table
	Delivery Date	Char(8)		YYYYMMDD format
	Seal id	Char(15)		Customs defined id for the container's seal
	Freight Type	Char(6)		Code that identifies the container type. Validated against the FREIGHT_TYPE table.
	Freight Size	Char(6)		Code that identifies the container size. Validated against the FREIGHT_SIZE table.
	In Transit No.	Char(15)		External transit number
	In Transit Date	Char(8)		YYYYMMDD format
		Char(15)		Identifies the Bill of Lading or Air Way Bill
	Candidate Ind	Char(1)	Defaulted to 'N'	Identifies a complete Transportation record. Valid values are 'Y' and 'N'
DPOIT	Record descriptor	Char(5)	DPOIT	Order/Item detail info
	Line id	Char(10)		Unique file line id

Record Name	Field Name	Field Type	Default Value	Description
	ACD_Code	Char(1)		Determines which process to perform 'A'dd, 'C'hange, 'D'elete.
	<i>Rush Ind</i>	<i>Char(1)</i>	<i>Defaulted to 'N'</i>	<i>Identifies whether or not the item should be on a 'Rush' delivery. Valid values are 'Y' and 'N'</i>
	Order number	Number(8)		Internal Retek order no
	Item	Number(8)		Internal Retek Item (style, SKU or Pack)
	Invoice id	Char(30)		Identifies the Commercial Invoice
	Invoice date	Char(8)		YYYYMMDD format
	Currency Code	Char(3)		Currency that the Currency Amount is reported in. Validated against CURRENCIES table.
	Exchange Rate	Number(20)		The exchange rate back to the primary currency (10 implied decimals)
	Invoice amt	Number(20)		Amount charged by supplier for the PO/Item. (4 implied decimal places)
	Origin Country id	Char(3)		Identifies where the PO/Item was made
	Consolidation Country id	Char(3)		Identifies where the PO/Items were consolidated
	Export Country id	Char(3)		Identifies where the PO/Items were shipped from
	Status	Char(6)		Identifies the PO/Item status. Valid values are found in the TRCO Code Type on CODE_DETAIL
	Receipt ID	Char(30)		Identifies the external receipt number
	FCR id	Char(15)		Identifies the Freight Cargo Receipt id
	FCR date	Char(8)		YYYYMMDD format

Record Name	Field Name	Field Type	Default Value	Description
	Packing Method	Char(6)		Identifies the Packing Type (Hanging or Flat). Valid values are 'HANG' or 'FLAT'
	Lot Number	Char(15)		Identifies the Lot Number of the PO/Item
	Item Qty	Number(12)		Qty of Items (4 implied decimals)
	Item QTY UOM	Char(4)		Identifies the UOM associated with the item quantity
	Carton QTY	Number(12)		Qty of Cartons (4 implied decimals)
	Carton QTY UOM	Char(4)		Identifies the UOM associated with the carton quantity
	Gross WT	Number(12)		Gross weight (4 implied decimals)
	Gross WT UOM	Char(4)		Identifies the UOM associated with the gross weight
	Net WT	Number(12)		Net Weight (4 implied decimals)
	Net WT UOM	Char(4)		Identifies the UOM associated with the net weight
	Cubic	Number(12)		Cubic size (4 implied decimals)
	Cubic UOM	Char(4)		Identifies the UOM associated with the cubic size
	Comments	Char(256)		User Comments
FTAIL	Record type	Char(5)	FTAIL	
	Line id	Char(10)		Unique file line id
	#lines	Number(10)		Total number of transaction lines in file (not including FHEAD and FTAIL)

Stock Allocation Transfer Record Purge [tsfalprg]

Design Overview

The purpose of this module is to purge 'SA' transfer records deleting all rows from the transfer allocation, header and detail tables based on the number of months of transfer history to be retained. The number of transfer history months to be retained is specified on system options. If the difference in the number of months between today and the date on which the transfer was closed is greater than or equal to the number of transfer history months, the header and detail record are purged.

To keep the warehouse management system and the host system aligned, a file containing all of the deleted transfers will be created. These transfers can then be sent to the warehouse management system to be deleted as well. The file will contain the shipment number, Location (DC), and an action type of 'D'elele. The shipment number will correspond to the Nautilus' distribution number.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
TSFALLOC	No	Yes	No	No	Yes
TSFHEAD	No	Yes	No	No	Yes
TSFDETAIL	No	No	No	No	Yes
SHIPMENT	No	No	No	No	No
SHIPSKU	No	No	No	No	Yes

Scheduling Constraints

Processing Cycle: AD-HOC

Scheduling Diagram: After TSFPRG.PC

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

```

EXEC SQL DECLARE c_tsfalloc CURSOR FOR
SELECT tsfalloc.tsfalloc_no,
       ','||to_char(tsfalloc.tsfalloc_no)
FROM tsfalloc,
     v_restart_tsfalloc vrt
WHERE NOT EXISTS (SELECT 'x'
                  FROM tsfhead
                  WHERE tsfhead.tsfalloc_no = tsfalloc.tsfalloc_no
                  AND tsfhead.status not in ('M','C','D')
                  OR MONTHS BETWEEN(tsfhead.close_date, :vdate) <
:tsf_history_months)
AND tsfalloc.tsfalloc_no = vrt.driver_value
AND vrt.driver_value = :ora_restart_driver_value
AND vrt.num_threads = :ora_restart_num_threads
AND vrt.thread_val = :ora_restart_thread_val
AND tsfalloc.tsfalloc_no > nvl(:ora_restart_tsfalloc, -999)
ORDER BY tsfalloc.tsfalloc_no;

```

Function Level Description

```

init()
Declare array for tsfalloc_no
Declare array for tsf_no
Declare array for shipment
Define array size
Initialize restart arrays – call restart_init()

```

Select vdate from period.

Select tsf_history_months from system_options.

process()

(see program hstwksst.pc)

initialize tsfalloc_rec_ctr = 0

tsfalloc_total_rec = 0

tsfhead_rec_ctr = 0

tsfhead_total_rec = 0

shipment_rec_ctr = 0

shipment_total_rec = 0

tsfalloc_flag = 1

tsfhead_flag = 1

shipment_flag = 1

open driving cursor

while

if !(tsfalloc_flag) break

- Fetch the records into an array of tsfalloc numbers.

EXEC SQL FOR :num fetch into tsfalloc array

- When no more records are found, set a flag to indicate that the loop should be exited at the end of the process

```
if (not found) tsfalloc_flag = 0;
```

- Determine the number of records processed this time through the loop. This is used to determine how much of the array was used up

```
tsfalloc_rec_ctr = NUM_RECORDS_PROCESSED –
```

```
tsfalloc_total_rec;
```

```
tsfalloc_total_rec += tsfalloc_rec_ctr;
```

```
for i = 0..tsfalloc_rec_ctr loop
```

```
    open cursor fetching all transfer numbers for the tsfalloc
```

```
number
```

```
    while(1)
```

```
        - Fetch tsfhead numbers into an array based on the
```

```
current tsfalloc number.
```

```
        for :num fetch into tsfhead array
```

- When no more records are found, set a flag to indicate that the loop should be exited at the end of the process

```
if (not found) tsfhead_flag = 0;
```

- Determine how many records were processed this time through the loop. This is used to determine how much of the array was used up

```
tsfhead_rec_ctr = NUM_RECORDS_PROCESSED –
```

```
tsfhead_total_rec;
```

```
tsfhead_total_rec += tsfhead_rec_ctr;
```

```

                                for i = 0..tsfhead_rec_ctr loop
                                    open cursor fetching all shipment numbers
                                for the transfer number

                                    while(1)
                                        - Fetch shipment numbers into an array based on the current tsfhead number
                                            for :num fetch into shipment
                                                array

                                        - When no more records are found, set a flag to indicate that the loop should be
                                          exited at the end of the process
                                            if (not found) shipment_flag = 0;

                                        - Determine the number of records processed this time through the loop
                                            shipment_rec_ctr =
NUM_RECORDS_PROCESSED -
                                            shipment_total_rec;

                                            shipment_total_rec +=
                                            shipment_rec_ctr;

                                            if !(shipment flag) break;
                                        end while

                                        - Delete from shipsku and shipment for all
                                          of the shipment numbers in the array
                                            EXEC SQL FOR :shipment_rec_ctr
                                                delete from shipsku where shipment =
:shipment;
```

```
EXEC SQL FOR :shipment_rec_ctr
    delete from shipment where shipment =
:shipment;

if the to or from location type is a warehouse
    write_detail_to_file()
end if;

end for

if !(tsfhead flag) break;
end while

- Delete from tsfdetail and tsfhead for all of the transfer
numbers in the array

EXEC SQL FOR :tsfhead_rec_ctr
    delete from tsfdetail where tsf_no = :tsf_no;

EXEC SQL FOR :tsfhead_rec_ctr
    delete from tsfhead where tsf_no = :tsf_no;

end for

- Delete from tsfalloc for all tsfalloc numbers in the array.
EXEC SQL FOR :tsfalloc_rec_ctr
    delete from tsfalloc where tsfalloc_no = :tsfalloc_no;
```

```
if !(tsfalloc flag) break;
```

```
end while
```

```
close cursor
```

```
write_detail_to_file()
```

This function will write the 2 files to download to the warehouse management system. One for stock orders (ie. transfers from the warehouse) and another file for transfers coming into the warehouse (ie. PO). If the from location of the transfer is a warehouse, then a stock order record will be written to the stock order file. If the to location of the transfer is a warehouse, then a record is written to the PO file.

```
final()
```

```
Call restart_close().
```

File I/O

Stock Order File

File Header:

Field Name	Field Type	Default Value from RLS	Description
File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
File Line Sequence	Number(10)	specified by external system	Line number of the current file
File Type Definition	Char(4)	TSFD	Identified file as 'Inventory Adjustments'
File Create Date	Date	Sysdate	date file was written by external system

Transaction Header:

File Type Record Descriptor	Char(5)	THEAD	Identifies file record type
File Line Sequence	Number(10)	Specified by external system	Line number of the current file
Transaction Set Control Number	Number(14)	Specified by external system	used to force unique transaction check
Action Type	Char(1)	'D' (hardcode) +1 for space	'A'dd, 'D'elele, 'M'odify
Location (DC)	Number (4)	Tsfhead.from_loc +1 for space	Code for the DC (future use). This field will not be used by Logistics, however it will allow RMS to create a separate file for each warehouse.
Transaction Date/Time	YYYYMMDDHHMI	Period.vdate +1 for space	Date/Time created in RMS
Distribution Number	Char(9)	Shipment.shipment +1 for space	Unique identifier of the distribution.
		408 spaces	Do not need to send whole file. Just the shipment number, action type, and Location(DC), which will be used for sorting by warehouse in the conversion program.

Transaction Tail:

File type record descriptor	Char(5)	TTAIL	Identifies file record type
File Line sequence	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 Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
File Line Sequence	Number(10)	specified by external system	Line number of the current file
File Line Count	Number(10)	total detail + transaction head lines	sum of all transaction lines, not including file header and trailer

Transfers to the DC file (ie. PO):

File Header:

Nautilus – Field Name	Field Type	Default Value	Description
File Type Descriptor	Char(5)	FHEAD	Identifies file record type
File Line Identifier	Number(10)	Nine leading zeroes: 0000000001	ID of current line being processed by input file.
File Type Definition	Char(4)	TDCH	Identifies file as ‘DC Header download’
File Create Date	Date	create date	date file was written by external system

Transaction Detail

File Type Descriptor	Char(5)	FDETL	Identifies file record type
File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
Action Type	Char (1)	default to ‘D’ + 1 for space	‘A’dd, ‘D’elete, ‘M’odify
Location (DC)	NUMBER (4)	shipment.location + 1 for space	Code for the DC (future use)
Transaction Date/Time	YYYYMMDDHHMM	Sysdate + 1 for space	Date/Time created on host
PO Number	VARCHAR (9)	shipment + 1 for space	Unique identifier of the PO.
		48 spaces	Do not need to send whole file. Just the shipment number, action type, and Location(DC), which will be used for sorting by warehouse in the conversion program.

File Trailer:

File Type Descriptor	Char(5)	FTAIL	Identifies file record type
File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
File Record Counter	Number(10)	determined internally	Number of records/transactions processed in current file (only records between head & tail)

Technical Issues

N/A

Combine Warehouse Originated Transfers [tsfcomb]

Design Overview

The purpose of this program is to create transfers with a transfer type

Combined Transfer (tsfhead.transfer_type = 'CT'). Transfers from warehouses to stores which are scheduled to be shipped the following day or to other warehouses (no ship schedule is set up for warehouses) are eligible to be combined if:

- The transfer type = 'SR' (Store Requisition) OR
- The transfer type = 'PO' (Cross-dock PO) OR
- The transfer type = 'SA' (Stock Allocation), the merge_ind = 'Y' and the release_date <= today and Freight code is Normal or Expedite

After the transfers have been combined, 'PO' and 'SR' types are deleted, and the status of the 'SA' records are updated to 'M' (Released for Shipment) if all allocated quantity has been filled for this transfer.

The new transfer with a transfer type of 'CT' will be created in 'A' status.

Note: Transfers with a freight type = 'E' (Expedite) and a release date <= today will ignore the shipping schedule and be combined. Transfers with a freight type = 'H' (Hold) will be ignored by this program. Transfers with different routing types must be combined separately.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
TSFHEAD	Yes	Yes	Yes	Yes	Yes
TSFDETAIL	Yes	Yes	Yes	Yes	Yes
TSFALLOC	Yes	Yes	No	Yes	No
STORE_SHIP_SCHEDULE	Yes	Yes	No	No	No

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: This program must run before TSFDNLD.PC and after TSFRESV.PC

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: STORE/WH

Restart Recovery

```
SELECT th.tsf_no,
       th.from_loc,
       th.to_loc_type,
       th.to_loc,
       NVL(th.dept, -1),
       th.tsf_type,
       th.freight_code,
       th.routing_code,
       td.sku,
       NVL(td.tsf_qty, 0),
       th.rowid
FROM tsfhead th,
     tsfdetail td,
     tsfalloc ta,
     v_restart_store_wh rv
WHERE th.tsf_type      in ('SR', 'PO', 'SA')
   AND th.status       = 'A'
   AND th.from_loc_type = 'W'
   AND th.freight_code  in ('N', 'E')
   AND th.tsf_no        = td.tsf_no
   AND nvl(ta.merge_ind,'Y') = 'Y'
   AND th.tsfalloc_no   = ta.tsfalloc_no (+)
   AND nvl(ta.release_date, to_date(:ov_tomorrow,'YYYYMMDD'))
      <= to_date(:ov_tomorrow,'YYYYMMDD')
```

```
AND rv.driver_value    = th.to_loc  
AND rv.num_threads     = :ora_restart_num_threads  
AND rv.thread_val      = :ora_restart_thread_val  
AND th.tsf_no          > NVL(:ora_restart_tsf_no, -999)  
ORDER BY th.tsf_no;
```

Shared Modules

- Restart_init() - initialize restart recovery tables
- Restart_commit()
- Restart_close() - commit changes and update restart recovery tables

Function Level Description

Init()

declare arrays for updating TSFHEAD and deleting from TSFHEAD

initialize restart recovery – call restart_init

select vdate from period and determine tomorrow's date

Process()

initialize ship schedule flag = 1

initialize record counter = 0

open driving cursor

loop

fetch driving cursor into variables

if (not found) break

- A freight type of 'E' (expedite) will override the shipping schedule and ship tomorrow regardless of the to and from location types

- Only a 'W' to 'S' transfer uses the shipping schedule

if (freight type == 'N' && from_loc_type == 'W' && to_loc_type ==
'S')

return = validate_ship_schedule()

```
        if (return)
            ship schedule flag = 0
        else if (return < 0)
            Error
        end if
    end if

    if (ship schedule flag)
        post_transfers()
    end if
end loop
close driving cursor

update_delete_transfer()

validate_ship_schedule()

validate that a ship date exists tomorrow for the from warehouse/to store on
STORE_SHIP_DATE

if (error)
    return (-1)
else if (not found)
    return(1)
else - found
    return(0)
end if
```

write_recs_to_struct()

- Two arrays are necessary here because transfer with a type of 'SA' will be updated to 'M' status and transfers of type 'PO' and 'SR' will be deleted.

if (transfer type == 'SA')

 insert variable values into tsfhead_update array

else

 insert variable values into tsfhead_delete array

end if

increment counter

post_transfers()

- This function will either write new TSFHEAD/TSFDETAIL records or update existing records

retrieve a TSFHEAD record with the same department, from/to location, freight code and routing code with a transfer type = 'CT' and a status = 'A'

if (not found)

- insert TSFHEAD record with transfer type = 'CT' and status = 'A' using fetched values

(populate create id and approval id with 'BATCH')


```
- insert TSFDETAIL record for the 'CT' transfer using the fetched values
else
    retrieve a TSFDETAIL record for the item/pre-pack
    if (not found)
- insert TSFDETAIL record for the transfer using the fetched values
    else
        - update transfer quantity for the item/pre-pack
    end if
end if

write_recs_to_struct()

update_delete_transfer()

- delete from tsfdetail and tsfhead for all transfers in the tsfhead_delete array

if tsfdetail.allocated_qty = tsfdetail.fill_qty for all items on the transfer then
-- update tsfhead (set status = 'M' and close date = :vdate) for all transfers in the
tsfhead_update array
else
-- update tsfdetail.tsf_qty = 0 and set tsfhead.status = 'A'
end if

Final()
```

Exit Program – call restart_close()

Technical Issues

N/A

Transfer to DC Download [tsfdcdld]

Design Overview

This program downloads transfer to DC information for the warehouse to process. This program will write purchase order header and detail records for each warehouse shipment in input status from the shipment table.

The output files of this program will be in a Retek Standard flat. A conversion program will parse and convert the format of the flat files and create a file according to Nautilus' flat file specifications.

The tsfdcdld.pc needs to be modified to change the format of Unit Quantity, when downloading information to RDM. The new Unit Quantity field for the interface is now Number(12,4) opposed to the original Number(8).

Scheduling Constraints

Processing Cycle:	N/A
Scheduling Diagram:	Run in Phase 4 of batch schedule with ordrev.pc and tsfdnld.pc
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

The logical unit of work for this download module is a row from the shipment table, where the transfer number is not NULL, the location type is a 'W'arehouse and a status of 'I'input. The commit_max_ctr field on the restart_control table will determine the number of transactions that equal a logical unit of work.

The file records will be read in groups equal to the commit_max_ctr. After all records in a given read are processed, the restart commit logic and restart file writing logic will be called, and then the next group of file records will be read and processed. The commit logic will save the current file pointer position in the input file and commit all database transactions. The file writing logic will append the temporary holding files to the final output files.

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

Error handling will recognize two levels of record processing: process success and fatal errors. If a fatal error is returned, then file processing will end immediately. A restart will be initiated from the file pointer position saved in the restart_bookmark string at the time of the last commit point that was reached during file processing.

Driving cursor will be similar to:

```
SELECT s.shipment,
       ss.sku,
       ss.qty_expected,
       s.location,
       i.supplier,
       t.supp_pack_size,
       '||to_char(s.shipment)
FROM shipment s,
     shipsku ss,
     item_supplier i,
     tsfdetail t
WHERE s.shipment = ss.shipment
     AND t.sku(+) = ss.sku
     AND i.item(+) = ss.sku
     AND s.tsf_no is not NULL
     AND s.loc_type = 'W'
     AND s.status_code = 'I'
     AND t.tsf_no = s.tsf_no
     AND i.primary_supp_ind = 'Y'
     AND s.shipment > :os_restart_shipment
ORDER BY s.shipment;
```

Program Flow

N/A

Function Level Description

Init()

Initialize restart recovery.

Open output files, both Logistics' 'header' and 'detail' files.

Write FHEAD for both 'header' and 'detail' files.

Get the not after and not before dates from the codes table where code_type = 'DEFT'.

Process()

Loop through the records from the shipment table.

Call copy_variables()

The PO number will be populated using the shipment number.

Call write_FDETL_to_rms_head() for each new shipment number.

Call write_FDETL_To_rms_detail() for each sku in the shipment header.

If the shipment number changes, then

Call set_status()

End if;

Copy_variables()

Set the output file variables. If there is no primary supplier for the item, then supplier variable = 'NONE' and set the pack size variable to the transfer's pack size.

Write_FDETL_to_Rms_head()

Write the variables to the header output file

Write_FDETL_to_Rms_detail()

Write the variables to the detail output file

Set_status()

Update the shipment.status = 'E'xtracted.

Final()

Close restart logic

Close output file

The tsfdcl.d.pc needs to be modified to change the format of Unit Quantity, when downloading information to RDM. The new Unit Quantity field for the interface is now Number(12,4) opposed to the original Number(8).

I/O Specification

The output file should be accepted as a runtime parameter at the command line.

Transfer to DC Header file:

Record Name	Nautilus Field Name	Field Type	Default Value	Description
File Header	File Type Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	Nine leading zeroes: 0000000001	ID of current line being processed by input file.
	File Type Definition	Char(4)	TDCH	Identifies file as 'DC Header download'
	File Create Date	Date	create date	date file was written by external system
Transaction Detail	File Type Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
	Action Type	Char (1)	default to 'A'	'A'dd, 'D'elete, 'M'odify
	Location (DC)	NUMBER (4)	shipment.location	Code for the DC (future use)
	Transaction Date/Time	YYYYMMDDHHMI	Sysdate	Date/Time created on host
	PO Number	VARCHAR (9)	shipment	Unique identifier of the PO.

Record Name	Nautilus Field Name	Field Type	Default Value	Description
	Vendor Number	VARCHAR (7)	Primary supplier from item_supp_country table.	ID of the vendor that must exist in the RLS Vendor table
	Preassigned Flag	VARCHAR (1)	NULL	(future use)
	Deliver-Not-Before-Date	YYYYMMDD	Sysdate	Date before which the shipment should not be accepted
	Deliver-Not-After-Date	YYYYMMDD	Sysdate + days specified on code detail table	Date after which the shipment should not be accepted.
	Shipping Terms	VARCHAR2(3)	NULL	Code for shipping Terms
	Buyer Code	VARCHAR2(12)	NULL	Code for the PO Buyer
File Trailer	File Type Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
	File Record Counter	Number(10)	determined internally	Number of records/transactions processed in current file (only records between head & tail)

Transfer to DC detail file:

Record Name	Nautilus Field Name	Field Type	Default Value	Description
File Header	File Type Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	Ten zeroes: 0000000000	ID of current line being processed by input file.
	File Type Definition	Char(4)	TDCD	Identifies file as 'DC detail download'

Record Name	Nautilus Field Name	Field Type	Default Value	Description
	File Create Date	Date	create date	date file was written by external system
Transaction Detail	File Type Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
	Action Type	Char (1)	default to 'A'	'A'dd, 'D'elele, 'M'odify
	Location (DC)	NUMBER (4)	shipment.location	Code for the DC (future use)
	Transaction Date/Time	YYYYMMDDH HMI	sysdate	Date/Time created on host
	PO Number	VARCHAR (9)	shipment	Unique identifier of the PO.
	Item ID	VARCHAR (16)	shipsku.sku	Unique item identifier
	Requested Unit Qty	NUM(12,4)	shipsku.qty_expected	Number of units requested
	Ordered Casepack	NUM(12,4)	Tsfdetail.supp_pack_size	Expected number of units per carton
	Hang/Flat Indicator	CHAR(1)	hanger attribute or default door type from codes	F = Flat, H = Hang, A = All
File Trailer	File Type Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
	File Record Counter	Number(10)	determined internally	Number of records/transactions processed in current file (only records between head & tail)

Technical Issues

N/A

[tsfdcl.d.pc]

Design Overview

The tsfdcl.d.pc needs to be modified to change the format of Unit Quantity, when downloading information to RDM. The new Unit Quantity field for the interface is now Number(12,4) opposed to the original Number(8).

Scheduling Constraints

No change to RMS 8.0

Restart Recovery

No change.

Program Flow

No change.

Shared Modules

No change.

Function Level Description

The tsfdcl.d.pc needs to be modified to change the format of Unit Quantity, when downloading information to RDM. The new Unit Quantity field for the interface is now Number(12,4) opposed to the original Number(8).

I/O Specification

Transfer to DC Header file:

Record Name	Nautilus Field Name	Field Type	Default Value	Description
File Header	File Type Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	Nine leading zeroes: 0000000001	ID of current line being processed by input file.
	File Type Definition	Char(4)	TDCH	Identifies file as 'DC Header download'
	File Create Date	Date	create date	date file was written by external system
Transaction Detail	File Type Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
	Action Type	Char (1)	default to 'A'	'A'dd, 'D'elele, 'M'odify
	Location (DC)	NUMBER (4)	shipment.location	Code for the DC (future use)
	Transaction Date/Time	YYYYMMDDHHMI	Sysdate	Date/Time created on host
	PO Number	VARCHAR (9)	shipment	Unique identifier of the PO.
	Vendor Number	VARCHAR (7)	Primary supplier from item_supp_country table.	ID of the vendor that must exist in the RLS Vendor table
	Preassigned Flag	VARCHAR (1)	NULL	(future use)
	Deliver-Not-Before-Date	YYYYMMDD	Sysdate	Date before which the shipment should not be accepted

Record Name	Nautilus Field Name	Field Type	Default Value	Description
	Deliver-Not-After-Date	YYYYMMDD	Sysdate + days specified on code detail table	Date after which the shipment should not be accepted.
	Shipping Terms	VARCHAR2(3)	NULL	Code for shipping Terms
	Buyer Code	VARCHAR2(12)	NULL	Code for the PO Buyer
File Trailer	File Type Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
	File Record Counter	Number(10)	determined internally	Number of records/transactions processed in current file (only records between head & tail)

Transfer to DC detail file:

Record Name	Nautilus Field Name	Field Type	Default Value	Description
File Header	File Type Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	Ten zeroes: 0000000000	ID of current line being processed by input file.
	File Type Definition	Char(4)	TDCD	Identifies file as 'DC detail download'
	File Create Date	Date	create date	date file was written by external system
Transaction Detail	File Type Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
	Action Type	Char (1)	default to 'A'	'A'dd, 'D'eleate, 'M'odify

Record Name	Nautilus Field Name	Field Type	Default Value	Description
	Location (DC)	NUMBER (4)	shipment.location	Code for the DC (future use)
	Transaction Date/Time	YYYYMMDDH HMI	sysdate	Date/Time created on host
	PO Number	VARCHAR (9)	shipment	Unique identifier of the PO.
	Item ID	VARCHAR (16)	Shipsku.sku [AC1]	Unique item identifier
	Requested Unit Qty	Number 12,4	Shipsku.qty_expe cted [AC1]	Number of units requested
	Ordered Casepack	NUMBER(4)	Tsfdetail.supp_pa ck_size	Expected number of units per carton
	Hang/Flat Indicator	CHAR(1)	Hanger attribute or default door type from codes	F = Flat, H = Hang, A = All
File Trailer	File Type Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	File Record Counter	Number(10)	Determined Internally	Number of records/transactions processed in current file (only records between head & tail)

Technical Issues

N/A

Approved Warehouse Transfers Download [tsfdnld]

Function

This program processes all warehouse transfers that are approved, with a freight code of Normal or Expedite and have a release date equal to or less than tomorrow. If the destination location is a store, the store must be on the ship schedule to be shipped tomorrow. Shipments are created for these transfers and the shipment information is downloaded into a file to be used by an external WMS. Transfer status will be updated to 'E' (Extracted).

This program will produce two additional files. The first file contains component ticket and retail information, for non sellable pack items. This will provide the correct ticketing information for the warehouse to ticket the components of non sellable pack items. The second file contains outbound work order processing information for stock allocations. The work order information is found on the work order tables, wo_wip, wo_head, and wo_sku_loc.

When interfacing with Nautilus all three files will need to be converted into the proper flat file format, so that Nautilus can process.

Note: Transfers that are supposed to be combined into Combined Transfer (CT transfer type) will not be downloaded by this program. Transfers with a freight type = 'E' (Expedite) and a release date <= today will ignore the shipping schedule and be downloaded tonight. Transfers with a freight type = 'H' (Hold) will be ignored by this program.

Design Overview

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
TSFALLOC	Yes	Yes	No	No	No
TSFHEAD	Yes	Yes	Yes	Yes	No
TSFDETAIL	Yes	Yes	No	No	No
SHIPMENT	Yes	Yes	Yes	No	No
STORE_SHIP_DATE	Yes	Yes	No	No	No
WO_HEAD	Yes	Yes	No	No	No
WO_SKU_LOC	Yes	Yes	No	No	No
WO_WIP	Yes	Yes	No	No	No
ORDCUST	Yes	Yes	No	No	No
CUSTOMER	Yes	Yes	No	No	No
ITEM_TICKET	No	Yes	No	No	No
V_PACKSKU_QTY	No	Yes	No	No	No

The tsfdnld.pc needs to be modified to change the format of Unit Quantity, when downloading information to RDM. The new Unit Quantity field for the interface is now Number(12,4) opposed to the original Number(6).

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: Phase 3. Constraints: after TSFCOMB.PC

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

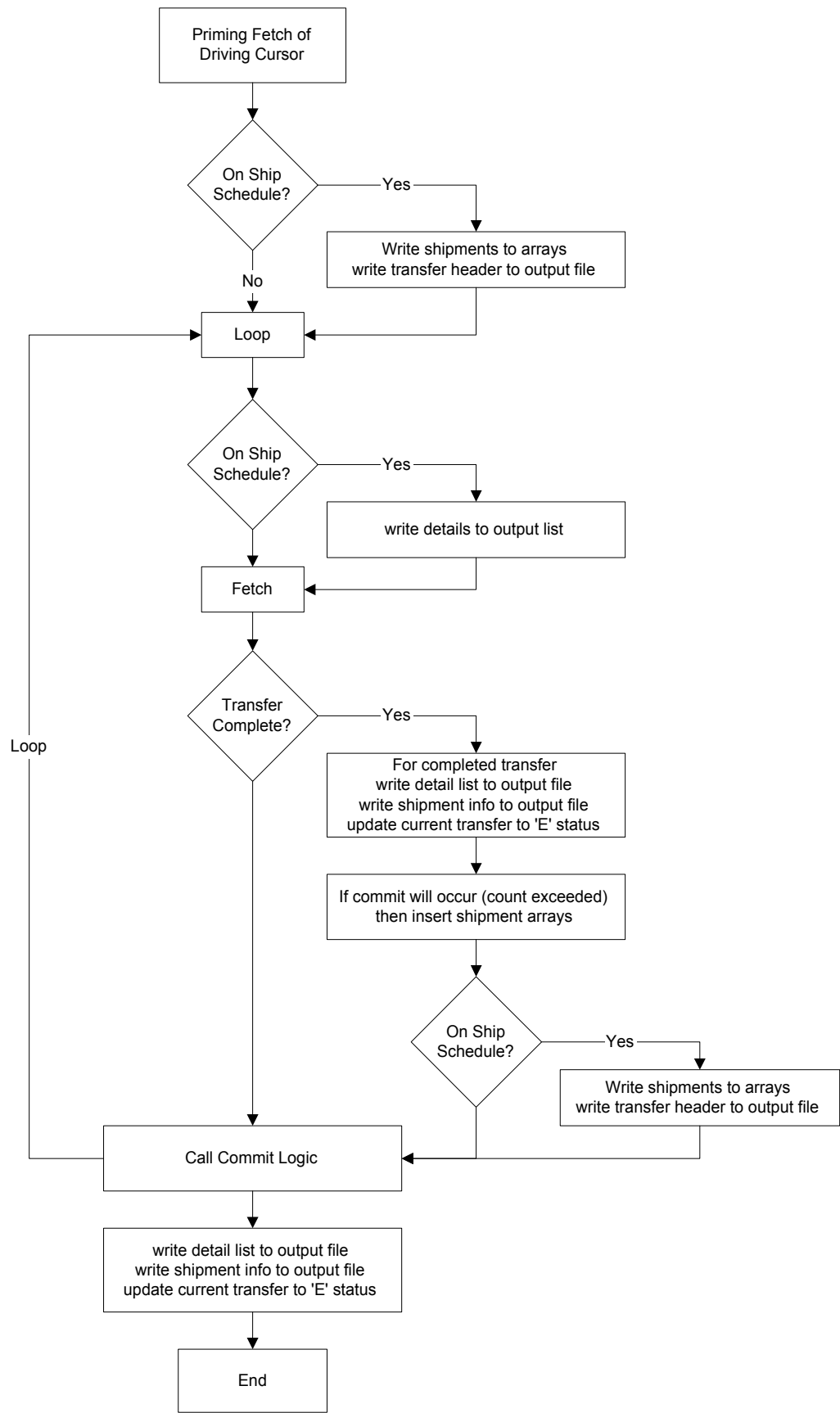
```
SELECT tsfhead.tsf_no,  
       tsfhead.from_loc_type,  
       tsfhead.from_loc,  
       tsfhead.to_loc_type,
```

```
tsfhead.to_loc,
tsfhead.tsf_type,
tsfhead.freight_code,
ROWIDTOCHAR(tsfhead.rowid),
'||to_char(tsfhead.tsf_no),
tsfdetail.sku,
(tsfdetail.tsf_qty)*1000,
nvl(tsfdetail.inv_status, 0)
FROM tsfhead,
tsfalloc,
tsfdetail
WHERE tsfhead.status = 'A'
AND tsfhead.freight_code in ('N','E')
AND tsfhead.from_loc_type = 'W'
AND tsfhead.tsf_type not in ('PO','SR')
AND nvl(tsfalloc.merge_ind,'N') = 'N'
AND tsfhead.tsfalloc_no = tsfalloc.tsfalloc_no (+)
AND nvl(tsfalloc.release_date, to_date(:ps_tomorrow,'YYYYMMDD'))
<= to_date(:ov_tomorrow,'YYYYMMDD')
AND tsfdetail.tsf_no = tsfhead.tsf_no
AND nvl(tsfdetail.tsf_qty,0) > 0
AND tsfhead.tsf_no > nvl(:ora_restart_tsf_no, -999)
ORDER BY tsfhead.tsf_no;
```

Shared Modules

NEXT_SHIPMENT_SQL used to get the next shipment number.

PRICING_ATTRIB_SQL.GET_RETAIL(): get the unit retail from item_zone_pricing table for a sku/store.



Program Flow

N/A

Function Level Description

Init()

Initialize restart recovery.

Open output file.

Format header, detail, and shipment buffers (for writing output).

Determine tomorrow's date

Determine order type

Call function get_order_type to determine order type

Call function to write output file header information, write_std_header()

Process()

This function should select all transfer details and quantities for transfers that are ready to ship from a warehouse tomorrow. Each transfer (header, detail information, and shipment information) should be written to an output file for the WMS to upload with transfer requirements. When a transfer has been completed, that is all information has been written to a file and the shipment information has been created, its status will be updated to Extracted ('E').

The flow of logic is as follows:

Fetch the first transfer record from the driving cursor.

Get_ship_flag (determines if current transfer is due to ship tomorrow)

if the transfer should be shipped then

call get_thead_info() to get the customer address information if it is a customer order type of transfer.

Call write_recs_to_struct() to create shipment number and write records to structure

Call write_head_to_str() to write to the THEAD structure.

End if;

Main processing loop through the transfer tables

If transfer should be shipped then

Call Get_detail_info() to get the ticketing and retail information. Also, decode the expedite flag.

Call write_detail_to_list() write TDETL to link list

Call Process_wo() to process the work order information

End if;

Fetch next transfer record

If the transfer number just changed, then

If the transfer should be shipped write into from the previous transfer to the file

Call Write_list_to_file() write link list of details to flat file.

Call Write_wo_to_file()

Call write_pack_to_file()

Call write_tail_to_file()

End if;

Call update_records() to update the appropriate tables

Now start working on the newly fetched transfer

Call `get_ship_flag()` to see if new transfer should be shipped

If transfer should be shipped, then

Call `Get_thead_info()`

Call `write_recs_to_struct()`

Call `write_head_to_str()`

End if;

End if;

Commit records and updates.

End of transfer loop

If the last transfer fetched should be shipped, then write final to file

Call `write_list_to_file()`

Call `write_wo_to_file()`

Call `write_pack_to_file()`

Call `write_tail_to_file()`

End if;

Call `update_records()`

`Get_ship_flag()`

This function calls `validate_ship_schedule()` to determine if transfer will be shipped tomorrow. If the transfer is set to expedite status, then the shipping schedule is ignored and the transfer is processed.

`validate_ship_schedule()`

This function validates that a ship date exists between today and tomorrow for the from warehouse and the to store combination (held on `STORE_SHIP_DATE` table).

get_thead_info()

This function retrieves the customer address from the customer table for the customer order transfer. If the customer is going to pick up the merchandise, then a message, “customer order for: < customer name > “ will be displayed in the event description. This will indicate to the warehouse that it is a customer order, pick up.

If customer order and ship direct

- set break by distro value = ‘Y’.
- populate billing and shipping addresses with customer address info.
- Set dest. Id = courier value from tsfhead
- Set Courier/route/service codes = NULL

If not customer order

- set break by distro value = ‘N’
- do not populate billing and shipping address
- set dest. Id = store or warehouse
- set courier/route/service codes = NULL

get_detail_info()

This function decodes the freight code

if freight_code = ‘E’ then

expedite_flag = ‘Y’;

else

expedite_flag = ‘N’;

end if;

Get the ticket type for the item from item_ticket table where the po_print_type = ‘R’ (ie. print at the time of receipt). There may be several ticket types for the item with ‘R’ print type, therefore, get the first ticket type in the fetch.

Get Unit retail for the item/location from the item_zone_price tables by calling the package PRICING_ATTRIB_SQL.GET_RETAIL.

If item is going to a store location call function comp_tckt() to write component ticketing file

process_wo ()

This function retrieves all the work order information for the selected stock allocation and Calls write_wo_to_list()

Write_wo_to_list()

This function writes the work order information to the structures

Write_wo_to_file()

This function prints out the work order structure to flat file

Comp_tckt()

This function selects from pack_head for the item and sellable_ind = 'N'.

If non Sellable 'P'ack item is found

loop through component items that make up the pack item on the v_packsku_qty table.

Call pricing_attrib_sql.get_retail package to get the retail for the component sku.

Call write_pack_to_list() Write FDETL record for component sku, retail, and ticket type to file

End loop;

end if;

write_pack_to_list()

This function writes the component ticketing and retail information to the structure

write_pack_to_file()

This function prints component ticketing and retail information structure to flat file

Write_std_header()

This function Increment counters and writes FHEAD record to file

Write_std_trailer()

This function increments counters and writes FTAIL record to file

`write_tail_to_file()`

This function writes the TTAIL structure to the output file

`write_detail_to_list()`

This function makes detail record string (TDETL) and add to linked list and calls `add_dtl_to_list()` function.

`add_dtl_to_list()`

This function will add ps_temp_dtl string to linked list

`get_order_type()`

This function gets order type from code_detail

`write_head_to_str()`

This function gets order header string (THEAD) and write structure

`Write_recs_to_struct()`

This function will be called when a new transfer number is encountered. Transfer header information is written to arrays that will update the status. A new shipment number is created and shipment information is written to arrays that will insert new shipment records into the shipment table.

`write_list_to_file()`

This function writes linked list detail records to file

`update_records()`

perform array update of tsfhead using rowid, set status = 'E'

perform array insert of newly created shipments

`Final()`

Call function to write output file trailer information, `write_std_trailer()`.

The tsfdnld.pc needs to be modified to change the format of Unit Quantity, when downloading information to RDM. The new Unit Quantity field for the interface is now Number(12,4) opposed to the original Number(6).

File I/O

Output files should be specified on the command line

Transfer download file

Record Name	Field Name	Field Type	FieldValue	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	TSFD	Identified file as 'Inventory Adjustments'
	File Create Date	Date	Sysdate	Date file was written by external system
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type
	File Line Sequence	Number(10)	Specified by external system	Line number of the current file
	Transaction Set Control Number	Number(14)	Specified by external system	Used to force unique transaction check
	Action Type	Char(1)	'A' (hardcode)	'A'dd, 'D'elele, 'M'odify
	Location (DC)	Number (4)	Tsfhead.fro m_loc	Code for the DC.
	Transaction Date/Time	YYYYMMDDHHMI	Period.vdate	Date/Time created in RMS
	Distribution Number	Char(9)	Shipment.sh ipment	Unique identifier of the distribution.
	Download Comment	Char (30)	NULL	Comment to be printed on the label (for future use)
	Pick-Not-Before-date	YYYYMMDD	Period.vdate	Date before which merchandise will not be distributed

Record Name	Field Name	Field Type	FieldValue	Description
	Pick-Not-After-Date	YYYYMMDD	Period.vdate + (specified time from codes table)	Date by which merchandise must be distributed. Extra days will be determined by a code type = 'DATE'
	Event Code	Char(6)	NULL or tsfalloc.tsfalloc_no	Identifier of event. Only used for stock allocations
	Event Description	Char(25)	NULL or tsfalloc.allo c_desc	Description of event. Only used for stock allocations
	Priority	Char(4)	Default to 1	Priority 1=highest
	Order Type	Char(9)	Default from system option Tables	Order type (Automatic, Manual or Wave)
	Break by Distro	Char(1)	Default from codes tables	Controls the mixing of orders (distros) in a container
	Carrier Code	Char(1)	NULL	Code of the carrier for the order
	Carrier Service Code	Char(6)	NULL	Carrier's service code for the delivery, First Class, etc.
	Route	Char(10)	NULL	Route specified for the delivery
	Ship Address Description	Char(30)	NULL or customer address	Used to store only customer order (ship direct) addresses.
	Ship Address line 1	Char(30)	NULL or customer address	Shipping address line 1. Used to store only customer order (ship direct) addresses.
	Ship Address line 2	Char(30)	NULL or customer address	Shipping address line 2. Used to store only customer order (ship direct) addresses.
	City	Char(25)	NULL or customer address	Shipping city. Used to store only customer order (ship direct) addresses.
	State	Char(3)	NULL or customer address	Shipping state. Used to store only customer order (ship direct) addresses.

Record Name	Field Name	Field Type	FieldValue	Description
	Zip	Char(10)	NULL or customer address	Shipping zip. Used to store only customer order (ship direct) addresses.
	Billing Address Description	Char(30)	NULL or customer address	The description (such as company name, etc.). This is the first line of the address block. Used to store only customer order (ship direct) addresses.
	Billing Address line 1	Char(30)	NULL or customer address	Billing address line 1. Used to store only customer order (ship direct) addresses.
	Billing Address line 2	Char(30)	NULL or customer address	Billing address line 2, Used to store only customer order (ship direct) addresses.
	Billing Address line 3	Char(30)	NULL or customer address	Billing address line 3, Used to store only customer order (ship direct) addresses.
	Amount 1	Number(8, 2)	NULL	Amount charge 1
	Amount 2	Number(8, 2)	NULL	Amount charge 2
	Amount 3	Number(8, 2)	NULL	Amount charge 3
	Order No.	Char(9)	NULL	Purchase Order Identifier
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence	Number(10)	Specified by external system	Line number of the current file
	Transaction Set Control Number	Number(14)	Specified by external system	used to force unique transaction check
	Action Type	Char(1)	'A' (hardcode)	'A'dd, 'D'elele, 'M'odify
	Location (DC)	Number (4)	NULL	Code for the DC (future use)
	Transaction Date/Time	YYYYMMDDHHMI	Period.vdate	Date/Time created in RMS

Record Name	Field Name	Field Type	FieldValue	Description
	Distribution Number	Char(9)	Shipment.shipment	Unique identifier of the distribution.
	Item ID	Char(16)	Tsfdetail.sku	Item identifier
	Requested Unit Qty	Num(12,4)	Tsfdetail.tsfdetail_qty	Number of units to distribute to the destination
	Destination ID	Number (4)	Tshead.routing_code (if ship direct to Customer order) Tshead.to_location (if store or wh)	Identifier of shipping destination. If customer order and ship direct, then field contains a carrier value. If it is direct to store or warehouse, then populate with the store or warehouse location.
	Price	Number (7,2)	Item_zone_price.unit_retail	Price of merchandise
	Print UPC Flag ('Y', 'N')	Char(1)	'N' (hardcode)	Whether to print UPC on tickets (Future use)
	Ticket Type	Number (4)	Item_ticket.ticket_type	Type of ticket refers to ticket type table. This field will be populate with the "ticket at receipt".
	Priority	NUMBER (4)	1 (hardcode)	Priority 1 = highest
	Expedite Flag	VARCHAR(1) 'Y' or 'N'	Tshead.freight_code (translate value to 'Y' or 'N')	Flag indicating whether the order should be shipped via normal or expedited carrier service
Transaction Trailer	File type record descriptor	Char(5)	TTAIL	Identifies file record type
	File Line sequence	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

Record Name	Field Name	Field Type	FieldValue	Description
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Line Count	Number(10)	total detail + transaction head lines	sum of all transaction lines, not including file header and trailer

Work Order Download File:

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Number(10)	Ten zeroes: 0000000000	ID of current line being processed by input file.
	File Type Definition	Char(4)	OWOD	Identifies file as 'Outbound Work Order Download'
	File Create Date	Date	Create date	date file was written by external system
Transaction Detail	File Type Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	Action Type	Char(1)	'A'	The action being performed on the work order. This will always be 'A' since transfer work orders can't be modified once they've been extracted.
	Location (DC)	Char(4)	Wo_sku_loc.wh	When an item is crossdocked, this field holds the value of the flow-through warehouse. Otherwise it holds the value of the final destination.
	Transaction Date/Time	Char (12) format: YYYYMMDDHHMI	Vdate	sysdate without the seconds

Record Name	Field Name	Field Type	Default Value	Description
	Distribution Number	Char(9)	Shipment	This field will hold the shipment number of the transfer the work order is associated with.
	Item ID	Char (16)	Wo_sku_loc.sk u	Valid item identifier for a staple SKU, fashion SKU, or Pack Item
	Dest ID	Number(4)	Wo_sku_loc.loc ation	Unique identifier of the final shipping destination.
	WIP Sequence No.	Number(7)	Wo_wip.seq_no	Work in Progress Sequence Number
	WO WIP Code	Char (6)	Wo_wip.code	WWIP code from codes table
File Trailer	File Type Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	Incremented internally	ID of current line being processed by input file.
	File Record Counter	Number(10)	Determined Internally	Number of records/transactions processed in current file (only records between head & tail)

Component ticketing file:

Component Ticketing file layout:

Record Name	Record	Default value	Field type	Description
File Header	File Line identifier	FHEAD	Char(5)	Identifies the trailer line
	Line number	0000000001	Number(10)	identifies file line number
	Program descriptor	CPTT	Char(4)	identifies the program
	Create date	YYYYMMDDHH24MISS	Char(14)	file create date
File detail	file record descriptor	FDETL	Char(5)	Detail line descriptor
	line number	Incremented internally	Number(10)	sequential line number
	Action_type	'A'	Char(1)	'A'dd, 'M'odify, 'D'elete
	Location	Tsfhead.from_loc	Number(4)	location that items will be transferred from
	Transaction date/time	vdate	Datetime(12)	date/time created in RMS
	distribution number	Shipment.shipment	char(9)	Unique identifier of the distribution.
	Master item id	Tsfdetail.sku	Char(16)	Unique identifier of the pack item
	Dest Id	Tsfdetail.to_loc	Number(4)	Identifier of the ship destination
	Component Item ID	v_packsku_qty.sku	Char (16)	item identifier of the component
	price	Item_zone_price.unit_retail	Number(7,2)	Price of the merchandise.
File Trailer	file record identification	FTAIL	Char(5)	File trailer
	line number	Incremented internally	Number(10)	sequential line number
	number of transaction lines	Total number of detail lines	Number(6)	total number of transaction lines in file (not including FHEAD and FTAIL)

Note: There is a space between fields in the RLS flat file format, except for the standard Retek flat file information, such as file type descriptor, file line identifier, file record counts.

Technical Issues

N/A

Transfer Receipts Upload [tsfiupld]

Design Overview

The purpose of this batch module is to process transfer shipment receipt transactions from an external system. The transfer shipment transactions will provide feedback on existing transfers in the Retek system. Transfers and shipments are closed if the criteria for closing is met. The following functions will be performed for each received item :

- update received quantity of transfer and shipment detail records
- update perpetual inventory and in transit qtys for destination location
- update in transit qtys for the source location
- update stock count's snapshot on hand quantity for destination location if stock count is in progress
- update unavailable inventory status quantity for NS (Non-salable) type of transfer for destination location
- update perpetual inventory with adjustments for detailed receipt discrepancies and create stock ledger stock adjustment transactions, if system_options.auto_close_tsf = 'Y'
- insert transfer and shipment detail records and perform transfer out logic if the received item does not exist on transfer detail table

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
TSFHEAD	No	Yes	Yes	Yes	No
TSFDETAIL	No	Yes	Yes	Yes	No
SHIPMENT	No	Yes	No	Yes	No
SHIPSKU	No	Yes	No	Yes	No
POS_MODS	No	No	Yes	No	No
PRICE_HIST	No	No	Yes	No	No
RAG_SKU_ST	No	Yes	No	Yes	No
WIN_STORE	No	Yes	No	Yes	No
RAG_SKUS_ST	No	Yes	No	Yes	No
WIN_WH	No	Yes	No	Yes	No
TRAN_DATA	No	No	Yes	No	No

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
RAG_SKUS	No	Yes	No	No	No
RAG_STYLE_ST	No	Yes	No	No	No
RAG_STYLE_WH	No	Yes	No	No	No
INV_STATUS_QTY	No	Yes	No	Yes	No
INV_STATUS_TYPE	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE 2 (daily)

Scheduling Diagram: This program must run after the transfer parse

module and will likely be run at the beginning of the batch run during the POS polling cycle, or possibly at the end of the batch run if pending warehouse transactions. It can be scheduled to run multiple times throughout the day, as WMS or POS data becomes available.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: STORE and WH
Threads driven by distinct files to process

Restart Recovery

The logical unit of work for the transfer in module is the complete feedback of a transfer shipment. This transfer shipment transaction will be defined as the logical unit of work. If any portion of the processing for the complete transfer shipment transaction fails, the entire transfer shipment must be re-processed.

A savepoint will be issued prior to processing a new transfer. If any record within the transaction fails, the whole transaction will be rolled back to the most recent savepoint. This way, the successfully processed transactions will remain posted to the database but not yet committed.

To prevent excessive rollback space usage, intermittent commits will be issued based on a commit counter. The recommended commit counter setting is 10000 records (subject to change based on experimentation). The commit counter is based on actual records processed, not overall transactions, nor the number of writes to the database, since the database interactions will be a constant multiplier of the commit counter. A transfer transaction cannot be committed to the database until it is complete so the commit counter is viewed as a minimum threshold, that once reached, will force a commit after the completion of the current transfer transaction.

Error handling will be based on the logical unit of work also. If a given record within a transfer transaction fails, that error will be posted to the standard error log for the batch module. If the error is of a non-fatal type, all subsequent detail records within the transfer will continue to be processed and any errors noted will continue to be posted. After processing all errors for the transaction, the entire transfer will be rejected to a runtime specified rejection file. If a fatal error is encountered, the file pointer at the time of the last commit will have been posted to the bookmark and all transactions from the last commit will be rolled back. Processing will commence with from the saved file position.

Shared Modules

TRANSFER_IN_SQL.EXECUTE: Package referenced to perform all transfer in logic, including

- update perpetual inventory – in transit values for destination location
- update stock count's snapshot on hand quantity for destination location if stock count is in progress
- update unavailable inventory status quantity for NS (Non-salable) type of transfer for destination location
- update perpetual inventory with adjustments for detailed receipt discrepancies and create stock ledger stock adjustment transactions, if system_options.auto_close_tsf = 'Y'

TRANSFER_IN_SQL.CLOSE_TSF_SHIP : Package referenced to close transfer and shipment. The criteria for closing are as follows :

If system_options.auto_close_tsf = 'Y'

The I_upd_intran_ind should be sent in as 'N' if the sku did not exist on the transfer. It should be 'Y' if the sku was on the transfer. That is, if the location was expecting the item it needs to update the in transit tables. If it was not expected the in transit tables should not be updated.

TRANSFER_OUT_SQL.EXECUTE: Package referenced to perform all transfer out logic, including

- create/update bill of lading
- create/update warehouse issues history (if transfer from a wh to a store)
- update perpetual inventory – in transit values
- update average cost of destination location
- create financial transactions for both the transfer out and the transfer in
- update stock count snapshots for cycle counts that are in progress

The I_upd_intran_ind should always be sent as 'N'. The package is only called in the sku is not expected and thus it should not update the in transit qtys. The following are called from TRANSFER_OUT_SQL and/or TRANSFER_IN_SQL packages and are thus, indirect calls.

STOCK_LEDGER_SQL.TRAN_DATA_INSERT: Package referenced by TRANSFER_OUT_SQL.EXECUTE to perform the stock ledger transaction inserts for the transfer out of the goods from the source location and the transfer in of the goods at the destination location.

NEW_ITEM_LOC: These stored procedures are used to create item/location relationships for locations that are to receive goods on a transfer and have not yet stocked the given item.

INVADJ_SQL.ADJ_UNAVAILABLE : called to update the unavailable inventory status quantity

INVADJ_SQL.ADJ_TRAN_DATA : called to write tran_data record for unavailable inventory adjustment

Function Level Description

init()

declare structure arrays for tsfdetail & shipsku

initialize restart recovery

select vdate from period and dept_level_transfers and auto_close_tsf indicators from system_options

open input file (transfer in)

- file should be specified as input parameter to program

open reject file (as a temporary file for restart)

- file should be specified as input parameter to program

set application image array - save the line counter

read file header record

process()

loop

read record from input file

if ('FTAIL') Exit Loop

if ('THEAD')

set savepoint and save current file pointer position

validate_THEAD()

process_THEAD()

reset detail count

end if

loop

read record from input file

if ('TTAIL') Exit Loop

if ('TDETL')

validate_TDETL()

```
process_TDETL()
write detail transaction to tsfdetail and shipsku array
end if

    if ( detail count = max array count )
        resize array structures for tsfdetail and shipsku
        increase max array count
    end if
    increment detail count
end loop

if ( no errors )
    post_transfers()
end if

    if ( non Fatal Error Encountered )
        reject_record - call write error and pass file pointer as of last savepoint and
        current file pointer
        Rollback transaction
    end if

    if ( transaction count > max commit count )
        restart file commit
        - save the current input file pointer position
        - save the line counter in restart image
    end if
end loop
```

restart commit final

validate_THEAD()

format_header_fields()

if (shipment number is provided)

- validate that shipment exists within the Retek system for a transfer

if validation fails then

(non-fatal error)

end if

else

- validate that the external shipment number matches an ext_ship_no on shipment
and exists for a transfer (tsf_no is not null)

if external shipment number provided does not exist on shipment

(non-fatal error)

end if

end if

validate that the transfer, for which the shipment is applicable to the to location
specified

if validation fails then

(non-fatal error)

end if

process_THREAD()

Fetch transfer type, from location type, and from location from tsfhead based on the transfer number.

validate_TDETL()

if (Item Type = 'UPC')

 select SKU from upc_ean based on the UPC and supplement

 if (UPC does not exist)

 write non-Fatal Error (UPC not found)

 end if

else if (Item Type = 'SKU')

 SKU = item value

end if

get item system indicator, department, class and subclass for the sku

check that SKU exists on the transfer detail table

if does not exist on transfer detail table

 insert into tsfdetail table with ship_qty = tsf_qty= 0 and receive_qty = qty
received from input file

 insert into shipsku table with qty_expected=0 and qty_received = qty
received from input file

 call transfer_out_sql.execute to process transfer out logic

else

 fetch nvl(qty received, 0) and qty expected from shipsku

if the qty received for the shipment is greater than zero (indicating that the shipment has been received at

least once) and this qty received plus the received qty from the input file is greater than the qty expected

on shipsku then

non-fatal error

else

update quantity received on both tsfdetail and shipsku tables

end if

end if

process_TDETL()

call TRANSFER_IN_SQL.EXECUTE package function

(see design specification for TRANSFER_IN_SQL)

post_transfers()

perform array update of tsfdetail and shipsku, adjust quantity received for both tables.

- call TRANSFER_IN_SQL.CLOSE_SHIP_TSF to determine whether or not the shipment and transfer can be closed based on the qty received. (see design specifications for TRANSFER_IN_SQL.CLOSE_SHIP_TSF)

format_header_fields()

assign input file fields to variables

if location type = 'ST'

set ora_to_type = 'S'

else if location type = 'WH'

set ora_to_type = 'W'

end if

-- ora_to_type will be passed to the TRANSFER_IN_SQL functions called from this batch program. These functions expect the type to be like TSFHEAD.TO_LOC_TYPE which uses 'S' to indicate a store and 'W' to indicate a warehouse.

format_detail_fields()

assign input file fields to variables

- because transfer quantity has an implied 4 decimal places

set transfer qty = transfer qty / 10000

ON Fatal Error

rollback to last physical commit point

Exit Program

ON Non-Fatal Error

rollback to last savepoint

write out complete transfer transaction to the reject file, pass file pointer at last savepoint and current file pointer

I/O Specification

Input File

The input file should be accepted as a runtime parameter at the command line.

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	TSFI	Identifies file as 'Transfer IN'
	File Create Date	Date	create date	date file was written by external system (YYYYMMDDHHMMS S)
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Number(14)	specified by external system	used to force unique transaction check

Record Name	Field Name	Field Type	Default Value	Description
	Transaction Date	Date	specified by external system	date the transfer was created in external system (YYYYMMDDHHMMS S)
	To Location Type	Char(2)	ST - store WH - warehouse	specifies the type of location receiving items
	To Location Value	Number(4)	Location identifier	specifies the receiving location id number
	Adjustment Flag	Char(1)	'Y' – Adjustment 'N' – Not Adjust.	Not used for Transfers – will always be 'N' (Used for receiver error adjustment for common receiving layout).
	Order Number	Number(8)	Retek order number	specifies the Retek order cross reference
	Shipment Number	Number(10)	Retek shipment number	specifies the Retek shipment cross-reference
	Supplier	Number(10)	Retek internal supplier number	file layout is common to Rcv.

Record Name	Field Name	Field Type	Default Value	Description
	External shipment number	Char(15)		specifies a shipment number from an external system
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Number(14)	specified by external system	used to force unique transaction check
	Detail Sequence Number	Number(6)	specified by external system	sequential number assigned to detail records within a transaction
	Item Type	Char(3)	UPC SKU VPN	item type will be represented as an UPC, SKU or VPN
	Item Value	Number(30)	item identifier	the id number of a SKU, UPC or VPN
	Supplement	Number(5)	supplemental identifier	used to further specify the id of an UPC item, or the pre-pack id reference
	Carton	Char(20)	Carton identifier	

Record Name	Field Name	Field Type	Default Value	Description
	Inventory Status	Number(2)	inventory status	used to indicate the type of non-salable merchandise transferred in an 'NS' transfer
	Receipt Quantity Sign Flag	Char(1)	'P' – positive 'N' – Negative	Always positive for transfer (file layout common to receiving which can have pos. or neg. adjustments)
	Received Quantity	Number(12,4)		Number of units received for transfer
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Sequence	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 Sequence	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 file header and trailer records

Reject File

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will need to be created by the transfer out module and a reject line counter will be required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and trailer record and the file line count will be equal to 0.

The reject filename should also be specified as a runtime parameter.

Error File

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

N/A

Transfer Shipments Upload [tsfoupld]

Design Overview

The purpose of this batch module is to accept transfer shipment details from an external system. The transfer transactions will provide feedback to existing transfers within the Retek system or initiate manual transfers created in an external system. The following functions will be performed for each transferred item :

- create/update transfer and shipment header and detail records.
- create item/location relation for receiving location (if it doesn't exist)
- update perpetual inventory and in transit qtys for source location
- update the average cost of item and in transit qtys for receiving location
- write financial transactions for both the transfer out and the transfer in
- update stock count's snapshot on hand quantity for source location and snapshot in transit quantity for destination location if stock count is in progress
- create/update bill of lading
- create/update warehouse issues history (if transfer from a wh to a store)
- update unavailable inventory status quantity for NS (Non-salable) type of transfer for source location
- update quantity transferred on allocation detail table if this transfer was created from standalone allocation

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
TSFHEAD	No	Yes	Yes	Yes	No
TSFDETAIL	No	Yes	Yes	Yes	No
SHIPMENT	No	Yes	Yes	Yes	No
SHIPSKU	No	Yes	Yes	Yes	No
POS_MODS	No	No	Yes	No	No
PRICE_HIST	No	No	Yes	No	No
RAG_SKUS_ST	No	Yes	No	Yes	No
WIN_STORE	No	Yes	No	Yes	No

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
RAG_SKUS_ST	No	Yes	No	Yes	No
WIN_WH	No	Yes	No	Yes	No
TRAN_DATA	No	No	Yes	No	No
RAG_SKUS	No	Yes	No	No	No
RAG_STYLE_ST	No	Yes	No	No	No
RAG_STYLE_WH	No	Yes	No	No	No
INV_STATUS_QTY	No	Yes	No	Yes	Yes
INV_STATUS_TYPES	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE 2 (daily)

Scheduling Diagram: This program must run before the transfer in batch module and will likely be run at the beginning of the batch run during the POS polling cycle, or possibly at the end of the batch run if pending warehouse transactions. It can be scheduled to run multiple times throughout the day, as WMS or POS data becomes available. In a true DC flow through type of operation, this program should also be run after Carton Receiving Upload (ctniupld) module to ship the cross-dock carton transfers created in ctniupld so that the goods received into DC for a cross-dock PO are shipped out to the final destination within the same day.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: STORE and WH
Threads driven by number of distinct files

Restart Recovery

The logical unit of work for the transfer out module is the discrete transfer transaction. Each transfer will be identified by the transfer number (if it already exists in the Retek system) or an unique transaction set number generated by the external system. This transfer transaction will be defined as the logical unit of work. If any portion of the processing for the complete transfer transaction fails, the entire transfer must be re-processed.

A savepoint will be issued prior to processing a new transfer. If any record within the transaction fails, the whole transaction will be rolled back to the most recent savepoint. This way, the successfully processed transactions will remain posted to the database but not yet committed.

To prevent excessive rollback space usage, intermittent commits will be issued based on a commit counter. The recommended commit counter setting is 10000 records (subject to change based on experimentation). The commit counter is based on actual records processed, not overall transactions, nor the number of writes to the database, since the database interactions will be a constant multiplier of the commit counter. A transfer transaction cannot be committed to the database until it is complete so the commit counter is viewed as a minimum threshold, that once reached, will force a commit after the completion of the current transfer transaction.

Error handling will be based on the logical unit of work also. If a given record within a transfer transaction fails, that error will be posted to the standard error log for the batch module. If the error is of a non-fatal type, all subsequent detail records within the transfer will continue to be processed and any errors noted will continue to be posted. After processing all errors for the transaction, the entire transfer will be rejected to a runtime specified rejection file. If a fatal error is encountered, the file pointer at the time of the last commit will have been posted to the bookmark and all transactions from the last commit will be rolled back. Processing will commence with from the saved file position.

Shared Modules

TRANSFER_OUT_SQL.EXECUTE: Package referenced to perform transfer out logic, including

- create item/location relation for receiving location (if it doesn't exist)
- update perpetual inventory for source location
- update the average cost of item for receiving location
- write financial transactions for both the transfer out and the transfer in
- update stock count's snapshot on hand quantity for source location and snapshot in transit quantity for destination location if stock count is in progress
- create/update bill of lading
- create/update warehouse issues history (if transfer from a wh to a store)
- update unavailable inventory status quantity for NS (Non-salable) type of transfer for source location
- update quantity transferred on allocation detail table if this transfer was created from standalone allocation

TRANSFER_IN_SQL.EXECUTE: Package referenced to perform transfer in logic for customer order types of transfers where the delivery type for the transfer is 'Ship Direct' :

- update perpetual inventory for destination location
- update stock count's snapshot on hand quantity for destination location if stock count is in progress
- update unavailable inventory status quantity for NS (Non-salable) type of transfer for destination location
- update perpetual inventory with adjustments for detailed receipt discrepancies and create stock ledger stock adjustment transactions, if system_options.auto_close_tsf = 'Y'

The following are called from TRANSFER_OUT_SQL and/or TRANSFER_IN_SQL packages and are thus, indirect calls.

STOCK_LEDGER_SQL.TRAN_DATA_INSERT: Package referenced by TRANSFER_OUT_SQL.EXECUTE to perform the stock ledger transaction inserts for the transfer out of the goods from the source location and the transfer in of the goods at the destination location.

NEW_STAPLE_LOC, NEW_FASHION_LOC, NEW_PACK_LOC: These stored procedures are used to create item/location relationships for locations that are to receive goods on a transfer and have not yet stocked the given item.

INVADJ_SQL.ADJ_UNAVAILABLE : called to update the unavailable inventory status quantity

INVADJ_SQL.ADJ_TRAN_DATA : called to write tran_data record for unavailable inventory adjustment

Function Level Description

init()

declare structure arrays for tsfdetail

initialize restart recovery

open input file (transfer out)

- file should be specified as input parameter to program

open reject file (as a temporary file for restart)

- file should be specified as input parameter to program

get vdate and department level transfer indicator from period table and system options

set application image array - save the line counter

read file header record

if (record type <> 'FHEAD') Fatal Error

process()

loop

read record from input file

if ('FTAIL')

Exit Loop

end if

if ('THEAD')

set savepoint and save current file pointer position

validate_THEAD()

reset detail count

process_THEAD()

end if

loop

check carton flag to determine if tdtl records will be for a carton or not

read record from input file (different structure for carton or regular)

if ('TTAIL') Exit Loop

if ('TDETL')

validate_TDETL()

process_TDETL()

end if

if (detail count = max array count)

resize array structures for tsfdetail

increase max array count

end if

increment detail count

end loop

if (no errors)

post_transfers() (don't call this if doing a carton)

end if

if (non Fatal Error Encountered)

reject_record - call write error and pass file pointer as of last savepoint and current file pointer

Rollback transaction

end if

if (transaction count > max commit count)

restart file commit

- save the current input file pointer position

- save the line counter in restart image

end if

end loop

restart commit final

validate_thead()

- validate transfer

-if external shipment number is 'CARTON', set carton flag and return from function

format_header_fields()

if (shipment number provided in transaction)

validate that the shipment number exists within Retek for a transfer. (check on shipment)

validate that the transfer within Retek has a status of 'A', 'E', 'S', 'C' (approved, extracted, shipped, closed) and is applicable to the

to/from locations specified (check on tsfhead) – also fetch transfer type

if shipment number provided does not exist on shipment in 'I', 'R' status for a transfer then

raise Non-Fatal Error

if transfer does not exist in Retek with the appropriate status and locations then

raise Non-fatal error

else if (no shipment number is provided)

if (external shipment number provided)

- validate to and from locations

if (loc_type = 'S')

check for existence on store table

else (loc_type = 'W')

check for existence on wh table

end if

if any location not exist, write non-Fatal error

- validate common transfer zone for store to store transfers

if (to_loc type = 'S' and from_loc = 'S')

check transfer zone - select transfer zone of the from location and the to location.

if (from_loc transfer zone $\neq$ to_loc transfer zone)

write non-Fatal Error (transfer zones incompatible)

end if

end if

else (no external shipment number)

All detail records must have a allocation number.

end if

end if

process_THREAD()

check for a bill of lading in 0 - open status for the destination location

retrieve the bill of lading number if one exists

if (bill of lading does not exist)

 get next bill of lading number

 insert bill of lading header (lad_head) record

end if

if bol number passed in ensure it is valid.

If it is not valid get next bol number.

if transfer type = 'CO'

 retrieve delivery type from the ORDCUST table

end if

validate_TDETL()

format_detail_fields()

if inventory status field is not blank, validate it against inv_status_types table

if no shipment / ext shipment in file

 every detail line must have an allocation.

if (shipment number in file)

```
        validate item exists on the transfer
    else
        if ( Item Type = 'UPC' )
            select sku from upc_ean based on the upc and supplement
            if ( upc does not exist )
                write non-Fatal Error ( upc not found )
            end if
        else if ( Item Type = 'SKU' )
            SKU = item value from the input file
            case ID = ' '
        end if
    end if
```

if the store rcv type is 'C' the carton field must be populated

- get item system indicator, department, class and subclass

```
if ( system indicator does not exist )
    write non-Fatal Error ( sku not found )
end if
```

process_TDETL()

The upd_resv_ind and the upd_intran_ind should be setup in the following way before calling transfer_out_sql.execute.

```
if :oi_new_tsf_flag = 1 then
```



```

if :os_store_rcv_type = 'A' then
    L_upd_resv_ind := 'N';
    L_upd_intran_ind := 'N';
else
    L_upd_resv_ind := 'N';
    L_upd_intran_ind := 'Y';
end if;

elsif :ora_tsf_type = 'CO' and :ora_deliver_type = 'S' or
      :os_store_rcv_type = 'A' then
    L_upd_resv_ind := 'Y';
    L_upd_intran_ind := 'N';
else
    if :os_tsf_status = 'C' then
        L_upd_resv_ind := 'N';
    else
        L_upd_resv_ind := 'Y';
    end if;
    L_upd_intran_ind := 'Y';
end if;

```

call TRANSFER_OUT_SQL.EXECUTE package function
 (see design specification for TRANSFER_OUT_SQL)

if transfer type = 'CO' and delivery type = 'S' or store receive type is 'A'
 call TRANSFER_IN_SQL.EXECUTE package function
 (see design specification for TRANSFER_IN_SQL)

write_recs_to_struct()

post_transfers()

if (shipment number was not passed in on the input file)

insert TSFHEAD (transfer_type = 'MR' or PO in an allocation is passed in,
ext_ref_no = external shipment number)

insert SHIPMENT (ext_ref_no_out should be the transaction control number,
ship date should be the transaction date)

perform array insert of TSFDETAIL

perform array insert of SHIPSKU

else (for all other Retek initiated transfer transactions)

try to update shipsku record if no data is found

perform array update of TSFDETAIL, set ship_qty – if transfer type = 'SA', set
tsf_qty = 0

perform array insert of SHIPSKU

- The this transfer is a customer order (tsf_type = 'CO') with a delivery type of
direct ship to customer, then this transfer must also be closed when it is sent.

if transfer type = 'CO' and delivery type = 'S' or store rcv type is 'A'

call TRANSFER_IN_SQL.CLOSE

(see design specification for TRANSFER_IN_SQL)

else if transfer type = 'SA' then

update TSFHEAD status to 'A' - approved

else

update TSFHEAD status to 'S' - shipped

end if

end if

format_header_fields()

assign input file fields to variables

if from location type = 'ST'

set ora_from_type = 'S'

else if from location type = 'WH'

set ora_from_type = 'W'

end if

if to location type = 'ST'

set ora_to_type = 'S'

else if to location type = 'WH'

set ora_to_type = 'W'

end if

format_detail_fields()

assign input file fields to variables

- transfer quantity has an implied 4 decimal places

$\text{transfer qty} = \text{transfer qty} / 10000$

process_carton()

Select details from transfer tables for the carton number; for each sku in the carton, call process_TDETL.

ON Fatal Error

- rollback to last physical commit point
- Exit Program

ON Non-Fatal Error

- rollback to last savepoint
- write out complete transfer transaction to the reject file, pass file pointer at last savepoint and current file pointer

I/O Specification

Input File

The input file should be accepted as a runtime parameter at the command line.

Important: The structure of the TDETL line will vary, depending on whether cartons are included or not. If cartons are included, the line will end after the item value field.

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	TSFO	Identifies file as 'Transfer OUT'
	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 Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Number(14)	specified by external system	used to force unique transaction check
	Transaction Date	Date	specified by external system	date the transfer was created in external system
	From Location Type	Char(2)	ST - store WH - warehouse	specifies the type of location sending items

Record Name	Field Name	Field Type	Default Value	Description
	From Location Value	Number(4)	location identifier	Specifies the sending location id number
	To Location Type	Char(2)	ST - store WH - warehouse	specifies the type of location receiving items
	To Location Value	Number(4)	location identifier	Specifies the receiving location id number
	Shipment Number	Number(10)	Retek shipment number	specifies the Retek shipment cross-reference
	External shipment	Char(15)	External shipment number	specifies external shipment number; will be CARTON when transferring cartons
	Courier	Char (20)	Courier used to ship order	
	Arrival date	Date	Arrival date	
	Number of boxes	Number(15)		Number of boxes in this transfer
	BOL number	Number(13)	Bill of lading	
Transaction Detail (Item)	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Number(14)	specified by external system	used to force unique transaction check

Record Name	Field Name	Field Type	Default Value	Description
	Detail Sequence Number	Number(6)	specified by external system	sequential number assigned to detail records within a transaction
	Item Type	Char(3)	UPC SKU	item type will be represented as a UPC or SKU
	Item Value	Number(13)	item identifier	the id number of a SKU or UPC
	Supplement	Number(5)	supplemental identifier	used to further specify the id of an UPC item
	Allocation Number	Char(6)	allocation identifier	Retek allocation number attached to the transfer
	Inventory Status	Number(2)	inventory status of item	used to indicate the type of non-salable merchandise transferred in an 'NS' transfer
	carton	Char(20)	carton identifier	UCC – 122 carton code
	Transfer Quantity	Number(12)		number of units to be transferred of the given item (*10000—4 implied decimal places)
Transaction Detail (Carton)	File Type Record Descriptor	Char(5)	TDETL	Identifies file record type

Record Name	Field Name	Field Type	Default Value	Description
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Transaction Set Control Number	Number(14)	specified by external system	used to force unique transaction check
	Detail Sequence Number	Number(6)	specified by external system	sequential number assigned to detail records within a transaction
	Item Type	Char(3)	CTN	item type will be represented as a CTN when transferring a carton
	Item Value	Char(20)	carton identifier	UCC – 122 carton code
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Sequence	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 Sequence	Number(10)	specified by external system	Current line number
	Number of transaction lines	Number(9)	specified by external system	total number of lines in file, excluding FHEAD and FTAIL

Output File

Record Name	Field Name	Field Type	Default Value	Description
	Record Type	Char (1)	H	Specifies file record type
	Store Order Number	Number (10)	Order No	Specifies store order number
	Division Type	Char (2)	Division Type	Specifies division type
	Warehouse	Number (6)	WH Loc	Specifies WH location value
	Store	Number (6)	Store Loc	Specifies ST location value
	Store Order Type	Number (4)	Store order type	Specifies store order type
	Store order comment	Char (255)	Comment	Specifies store order comment
	Ship Date	Number (14)	Ship date	Specifies date shipped

Detail

Record Name	Field Name	Field Type	Default Value	Description
	Record Type	Char (1)	D	Specifies record type
	Store Order number	Number (10)	Order No	Specifies Order Number
	Division type	Char (2)	SA, PO, MR, CO, AD	Specifies Division Type
	Xref Div Item	Number (8)		Specifies External Division Type
	UPC	Number (13)	UPC value	Specifies UPC Value
	UPC supplement	Number (5)	UPC supplement	Specifies UPC supplement value
	Unit of Measure	Char (2)	Unit of Measure	Specifies unit of measure
	SKU Deck Cost	Number (10)	Deck cost	Specifies SKU deck cost
	Quantity Shipped	Number (6)	Quantity Shipped	Specifies quantity shipped value

Reject File

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will need to be created by the transfer out module and a reject line counter will be required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and trailer record and the file line count will be equal to 0.

The reject filename should also be specified as a runtime parameter.

Error File

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

N/A

P. O. Receipt, Transfer Receipt, and Carton Receipt Parse [tsfparse]

Design Overview

This program will accept an input file containing receipt information for both purchase orders and transfers, as well as receipts processed with carton information. This program will then split out the PO receipts, the transfer receipts and the carton receipts into three files to be processed by rcvupld.pc, tsfiupld.pc, and ctniupld.pc. If the record in the input file contains an order number, then the record should be written to the rcvupld output file. If the order number field is filled with blanks, then the record should be written to the tsfiupld output file. If FDETL records exist, then the record should be written to the ctniupld output file. The program should also recalculate the total line count and file line identifier for each of the output files.

Scheduling Constraints

Processing Cycle: N/A

Scheduling Diagram: This program should always be run before rcvupld.pc, tsfiupld.pc, and ctniupld.pc

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Program Flow

N/A

Function Level Description

Main()

Parameters passed in:

- File name
- Logon ID/Password
- Input file name
- Receiving output file name
- Transfer output file name

init()

open input file (PO/Transfer)

file should be specified as input parameter to program

open output file (PO)

open output file (Transfer)

open output file(Cartons)

process()

This function will read all of the lines from the input file and write them to a transaction appropriate output file. PO transactions will be written to a file to be processed by rcvupld.pc. Transfer in records will be written to files to be processed by tsfiupld. and receipts with carton will be written to files to be processed by ctniupld.

The function will always use the read_next_line function to read a line from the input file into a character string buffer for processing. Validation will be done by separate functions according to the line type being processed (write_headers, process_head, write_tails). These validation functions will use the write_record function to write the line to the appropriate output file buffer.

The function should read and validate header information from the input file and write header information for each output file (via calls to write_headers function).

It will then loop through each transaction & should determine the output file based on the type of transaction that is been read (via process_head function). When the FTAIL segment is encountered, it will stop looping and write file trailer information (via write_tails function).

For each transaction it will loop through the detail records until the transaction tail record (TTAIL) is encountered. Each detail record will be validated and written to the appropriate output file by the process_detail function. Records to be written to the carton file will start with FDETL (and there will be no transaction head or tail records for cartons)

Read_next_line()

This function will read the next line from the input file and determine if a read error has occurred.

Write_headers()

This function will validate that the current line is a FHEAD type. The input file should have a file type definition of 'RCPT'. It will write the current buffer as the file header for the receiving output file. It will overwrite the 'RCPT' with 'TSFI' or 'CRTN' and write the newly formatted buffer to the transfer output file.

Process_head()

This function will validate that transaction header record. If the record type is FTAIL then return value to break from while loop in process function. Otherwise ensure that record type is THEAD.

If the order number is blank it is assumed to be a transfer and the "current" output file pointer should be set to the transfer file pointer, and the "current" line counter should be set to the transfer line counter. Otherwise, it is assumed to be a purchase order and the "current" output file pointer should be set to the receiving file pointer, and the "current" line counter should be set to the receiving line counter.

The function should call the write_records function to write the THEAD segment to the "current" file and increment the "current" line count.

Process_detail()

This function will validate the record type. If the record type is valid then the write_records function will be called to write the TDETL or TTAIL segment to the "current" file and increment the "current" record counter.

If the record type is TDETL then return normally, if the type is TTAIL then return a value that will break out of the while loop in the process function, and if it neither value then return a fatal error.

Write_tails()

This function will write a FTAIL record to each output file with the new line counts.

I/O Specification

Input File

The input file should be accepted as a runtime parameter at the command line.

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)	RCPT	Identifies file as 'Receipt' file
	File Create Date	Date	create date	Date file was written by external system (YYYYMMDDHHMMSS)
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	Number(14)	specified by external system	Used to force unique transaction check
	Transaction Date	Date	specified by external system	Date the transfer was created in external system (YYYYMMDDHHMMSS)
	Location Type	Char(2)	ST - store WH - warehouse	Specifies the type of receiving location
	Location Value	Number(4)	location identifier	Specifies the receiving location id number
	Adjustment Flag	Char(1)	'Y' – Adjustment 'N' – Not Adjust.	Not used for Transfers – will always be 'N' (Used for receiver error adjustment for common receiving layout).
	Order Number	Number(8)	Retek order number	Specifies the Retek order cross reference

Record Name	Field Name	Field Type	Default Value	Description
	Shipment Number	Number(10)	Retek shipment number	Specifies the Retek shipment cross reference
	Supplier Identifier	Number(10)	Retek supplier number	Specifies the Retek supplier cross reference
	Vendor ASN	Char(15)	vendor shipment number reference	Reference to vendor shipping document
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	Number(14)	specified by external system	used to force unique transaction check
	Detail Sequence Number	Number(6)	specified by external system	sequential number assigned to detail records within a transaction
	Item Type	Char(3)	UPC SKU VPN	item type will be represented as an UPC, a SKU, or a VPN
	Item Value	Number(30)	item identifier	the id number of a SKU, UPC or VPN
	Supplement	Number(5)	supplemental identifier	used to further specify the id of an UPC item
	Carton	Char(20)	carton number	unique identifier of each carton associated with a shipment.
	Inventory Status	Number(2)	inventory status of item	used to indicate the type of non-salable merchandise transferred in an 'NS' transfer
	Receipt Quantity Sign Flag	Char(1)	'P' – positive	Always positive for transfer (file layout common to receiving

Record Name	Field Name	Field Type	Default Value	Description
	Sign Flag		'N' – Negative	which can have pos. or neg. adjustments)
	Receipt Quantity	Number(12,4)		number of units received of the given item
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

Detail record for carton:

File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Carton Number	Char(20)	specified by external system	Specifies the unique UCC-128 Code for a carton
	Location	Number(4)	specified by external system	Location the carton was received by the external system
	Transaction Date	Date	specified by external system	Date the carton was received by the external system (YYYYMMDDHHMMSS)

Output files: tsfiupld.pc

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	TSFI	Identifies file as 'Transfer IN'
	File Create Date	Date	create date	date file was written by external system (YYYYMMDDHHM MSS)
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	Date	specified by external system	date the transfer was created in external system (YYYYMMDDHHM MSS)
	Location Type	Char(2)	ST - store WH - warehouse	specifies the type of receiving location
	Location Value	Number(4)	location identifier	specifies the receiving location id number
	Adjustment Flag	Char(1)	'Y' – Adjustment 'N' – Not Adjust.	Not used for Transfers – will always be 'N' (Used for receiver error adjustment for common receiving layout).
	Order Number	Char(8)	Retek order number	specifies the Retek order cross reference
	Shipment Number	Char(10)	Retek shipment number	specifies the Retek shipment cross reference

Record Name	Field Name	Field Type	Default Value	Description
	Supplier Identifier	Char(10)	Retek supplier number	specifies the Retek supplier cross reference
	External shipment number	Char(15)		specifies shipment number from 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
	Detail Sequence Number	Char(6)	specified by external system	sequential number assigned to detail records within a transaction
	Item Type	Char(3)	UPC SKU VPN	item type will be represented as an UPC, SKU or VPN
	Item Value	Char(30)	item identifier	the id number of a SKU, UPC or VPN
	Supplement	Char(5)	supplemental identifier	used to further specify the id of an UPC item
	Inventory status	Number(2)	inventory status	used to indicate the type of non-salable merchandise for a 'NS' transfer
	Receipt Quantity Sign Flag	Char(1)	'P' – positive 'N' – Negative	Always positive for transfer (file layout common to receiving which can have pos. or neg. adjustments)
	Receipt Quantity	Number(12,4)		number of units received of the given item
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

Record Name	Field Name	Field Type	Default Value	Description
	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 file header and trailer records

Output file: rcvupld.pc

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)	RCPT	Identifies file as 'Receipt'
	File Create Date	Date	create date	date file was written by external system (YYYYMMDDHHMSS)
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	Date	specified by external system	date the transfer was created in external system (YYYYMMDDHHMSS)
	Location Type	Char(2)	ST - store	specifies the type of receiving location

Record Name	Field Name	Field Type	Default Value	Description
			WH - warehouse	
	Location Value	Number(4)	location identifier	specifies the receiving location id number
	Adjustment Flag	Char(1)	'Y' – Adjustment 'N' – Not Adjust.	Not used for Transfers – will always be 'N' (Used for receiver error adjustment for common receiving layout).
	Order Number	Number(8)	Retek order number	specifies the Retek order cross reference
	Shipment Number	Number(10)	Retek shipment number	specifies the Retek shipment cross reference
	Supplier Identifier	Number(10)	Retek supplier number	specifies the Retek supplier cross reference
	Vendor ASN	Char(15)	vendor shipment number reference	reference to vendor shipping document
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
	Detail Sequence Number	Char(6)	specified by external system	sequential number assigned to detail records within a transaction
	Item Type	Char(3)	UPC SKU VPN	item type will be represented as an UPC, SKU or VPN
	Item Value	Char(30)	item identifier	the id number of a SKU, UPC or VPN

Record Name	Field Name	Field Type	Default Value	Description
	Supplement	Char(5)	supplemental identifier	used to further specify the id of an UPC item
	Carton	Char(20)	carton number	unique identifier of each carton associated with a shipment.
	Inventory status	number(2)	inventory status	used to indicate the type of non-salable merchandise for a 'NS' transfer
	Receipt Quantity Sign Flag	Char(1)	'P' – positive 'N' – Negative	Always positive for transfer (file layout common to receiving which can have pos. or neg. adjustments)
	Receipt Quantity	Char(12)		number of units received of the given item
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

Output file for ctniupld.pc (Note that this has only FDETL records, not THEAD, TDETL, and TTAIL)

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	File Type Definition	Char(4)	CRTN	Identifies file as 'Carton Receiving'
	File Create Date	Date	create date	date file was written by external system (YYYYMMDDHHMMSS)
File Detail	File Type Record Descriptor	Char(5)	FDETL	Identifies file record type
	File Line Sequence	Number(10)	specified by external system	Line number of the current file
	Carton Number	Char(20)	specified by external system	Specifies the unique UCC-128 Code for a carton
	Location	Number(4)	specified by external system	Location the carton was received by the external system
	Transaction Date	Date	specified by external system	date the carton was received by the external system (YYYYMMDDHHMMSS)
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Sequence	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

Reject File

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will need to be created by the receiving module and a reject line counter will be required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and trailer record and the file line count will be equal to 0.

The reject filename should also be specified as a runtime parameter.

Error File

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

N/A

Transfer Record Purge Transactions [tsfprg]

Design Overview

The purpose of this module is to purge transfer records, deleting all rows from the transfer header and detail table based on the number of months of transfer history to be retained. The number of transfer history months to be retained is specified on system options. If the difference in the number of months between today and the date on which the transfer was closed is greater than or equal to the number of transfer history months, the header and detail record are purged.

To keep the warehouse management system and the host system aligned, a file containing all of the deleted transfers will be created. These transfers can then be sent to the warehouse management system to be deleted as well. The file will contain the shipment number, Location (DC), and an action type of 'D'elele. The shipment number will correspond to the Nautilus' distribution number.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
PERIOD	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
TSFHEAD	No	Yes	No	No	Yes
TSFDETAIL	No	No	No	No	Yes
SHIPMENT	No	No	No	No	Yes
SHIPSKU	No	No	No	No	Yes
ORDCUST	No	No	No	No	Yes

Scheduling Constraints

Processing Cycle: PHASE AD-HOC

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A (single threaded)

Restart Recovery

```
SELECT tsfhead.tsf_no,  
       ','||to_char(tsfhead.tsf_no)  
FROM tsfhead,  
     v_restart_transfer vrt  
WHERE (tsfhead.status = 'D'  
       OR (tsfhead.status = 'C'  
           AND tsfhead.tsf_type <> 'SA'  
           AND MONTHS_BETWEEN(tsfhead.close_date, :vdate) >=  
:tsf_history_months))  
       AND vrt.driver_value = tsfhead.tsf_no  
       AND vrt.driver_value = :ora_restart_driver_value  
       AND vrt.num_threads = :ora_restart_number_threads  
       AND vrt.thread_val = :ora_restart_thread_val  
       AND tsfhead.tsf_no > nvl(:ora_restart_transfer, -999)  
ORDER BY tsfhead.tsf_no;
```

Function Level Description

init()

Call restart_init().

Initialize arrays

Fetch tsf_history_months from system_options and vdate from period.

process()

initialize tsfhead_rec_ctr = 0

tsfhead_total_rec = 0

shipment_rec_ctr = 0

shipment_total_rec = 0

tsfhead_flag = 1

shipment_flag = 1

open driving cursor

while

- Loop through the records in the driving cursor feeding them into an array as they are fetched.

for :num fetch into tsfhead array

-When no more records are found, set a flag to false so that the loop will be exited at the end of the processing.

if (not found) tsfhead_flag = 0;

- Determine the number of records processed this time through the loop.

tsfhead_rec_ctr = NUM_RECORDS_PROCESSED –

tsfhead_total_rec;

tsfhead_total_rec += tsfhead_rec_ctr;

- For all of the transfer numbers which were fetched into the array in the previous fetch, find the associated shipment numbers

for i = 0..tsfhead_rec_ctr loop

open cursor fetching all shipment numbers for the transfer number

while(1)

for :num fetch into shipment array

-When no more records are found, set a flag

to false so that the loop will be exited at

the end of the processing.

if (not found) shipment_flag = 0;

- Determine the number of records processed this time through the loop.

shipment_rec_ctr =

NUM_RECORDS_PROCESSED -

shipment_total_rec;

shipment_total_rec += shipment_rec_ctr;

if !(shipment flag) break;

- Delete from shipsku and shipment for all of the shipment numbers in the array

EXEC SQL FOR :shipment_rec_ctr

delete from shipsku where shipment =

:shipment;

EXEC SQL FOR :shipment_rec_ctr

delete from shipment where shipment =

:shipment;

if the to or from location type is a warehouse

```
write_detail_to_file()
```

```
end if;
```

```
end for
```

```
if !(tsfhead flag) break;
```

```
end while
```

- Delete from ordcust, tsfdetail, and tsfhead for all of the transfer number in the array.

```
EXEC SQL FOR :tsfhead_rec_ctr
```

```
delete from ordcust where tsf_no = :tsf_no;
```

```
EXEC SQL FOR :tsfhead_rec_ctr
```

```
delete from tsfdetail where tsf_no = :tsf_no;
```

```
EXEC SQL FOR :tsfhead_rec_ctr
```

```
delete from tsfhead where tsf_no = :tsf_no;
```

```
end for
```

```
end while
```

```
close cursor
```

```
write_detail_to_file()
```

This function will write the 2 files to download to the warehouse management system. One for stock orders (ie. transfers from the warehouse) and another file for transfers coming into the warehouse (ie. PO). If the from location of the transfer is a warehouse, then a stock order record will be written to the stock order file. If the to location of the transfer is a warehouse, then a record is written to the PO file.

```
final()
```

Call restart_close function

File I/O

Stock Order File

File Header:

Field Name	Field Type	Default Value from RLS	Description
File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
File Line Sequence	Number(10)	specified by external system	Line number of the current file
File Type Definition	Char(4)	TSFD	Identified file as 'Inventory Adjustments'
File Create Date	Date	Sysdate	date file was written by external system

Transaction Header:

File Type Record Descriptor	Char(5)	THEAD	Identifies file record type
File Line Sequence	Number(10)	Specified by external system	Line number of the current file
Transaction Set Control Number	Number(14)	Specified by external system	used to force unique transaction check
Action Type	Char(1)	'D' (hardcode) +1 for space	'A'dd, 'D'elele, 'M'odify
Location (DC)	Number (4)	Tsfhead.from_loc +1 for space	Code for the DC (future use). This field will not be used by Logistics, however it will allow RMS to create a separate file for each warehouse.
Transaction Date/Time	YYYYMMD DHHMI	Period.vdate +1 for space	Date/Time created in RMS

Distribution Number	Char(9)	Shipment.shipment +1 for space	Unique identifier of the distribution.
		408 spaces	Do not need to send whole file. Just the shipment number, action type, and Location(DC), which will be used for sorting by warehouse in the conversion program.

Transaction Tail:

File type record descriptor	Char(5)	TTAIL	Identifies file record type
File Line sequence	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 Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
File Line Sequence	Number(10)	specified by external system	Line number of the current file
File Line Count	Number(10)	total detail + transaction head lines	sum of all transaction lines, not including file header and trailer

Transfers to the DC file (ie. PO):

File Header:

Nautilus – Field Name	Field Type	Default Value	Description
File Type Descriptor	Char(5)	FHEAD	Identifies file record type
File Line Identifier	Number(10)	Nine leading zeroes: 0000000001	ID of current line being processed by input file.
File Type Definition	Char(4)	TDCH	Identifies file as ‘DC Header download’
File Create Date	Date	create date	date file was written by external system

Transaction Detail

File Type Descriptor	Char(5)	FDETL	Identifies file record type
File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
Action Type	Char (1)	default to ‘D’ + 1 for space	‘A’dd, ‘D’elete, ‘M’odify
Location (DC)	NUMBER (4)	shipment.location + 1 for space	Code for the DC (future use)
Transaction Date/Time	YYYYMMDDHHMM	Sysdate + 1 for space	Date/Time created on host
PO Number	VARCHAR (9)	shipment + 1 for space	Unique identifier of the PO.
		48 spaces	Do not need to send whole file. Just the shipment number, action type, and Location(DC), which will be used for sorting by warehouse in the conversion program.

File Trailer:

File Type Descriptor	Char(5)	FTAIL	Identifies file record type
File Line Identifier	Number(10)	incremented internally	ID of current line being processed by input file.
File Record Counter	Number(10)	determined internally	Number of records/transactions processed in current file (only records between head & tail)

Technical Issues

N/A

Reserve Inventory for Stock Allocation [tsfresv]

Function:

This program is used to reserve inventory for Stock Allocation transfers once its reserve date has arrived. Processing is done by selecting the items from TSFALLOC where the reserve date <= today, transfer qty is NULL or 0 and the allocated qty > filled qty for the item. This program will then check the available quantity at the source warehouse for the transfer and distribute the quantity to the locations in the allocation based on the allocation method specified. Pack size is taken into account. The fill quantity and transferred quantity for the allocation are updated with the amount of stock which will be filled at each location. Filled quantity will track the total qty filled throughout the life of the transfer. Transferred qty will track the qty which was filled since the last shipment for the SA transfer, or the last time the SA transfer is combined into CT transfer. Fill quantity should never exceed allocated quantity.

Design Overview

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
TSFHEAD	Yes	Yes	No	Yes	No
TSFDETAIL	Yes	Yes	No	Yes	No
WIN_WH	Yes	Yes	No	No	No
RAG_SKUS_WH	Yes	Yes	No	No	No
PACKWH	Yes	Yes	No	No	No
RAG_STYLE_WH	Yes	Yes	No	No	No

Scheduling Constraints

Processing Cycle:	N/A
Scheduling Diagram:	Run this program before TSFCOMB.PC
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	TSFALLOC

Restart Recovery

```

SELECT tsfalloc.alloc_method,
       tsfalloc.tsfalloc_no,
       tsfdetail.sku,
       SUM(NVL(tsfdetail.allocated_qty, 0) - NVL(tsfdetail.fill_qty, 0))
       sum_need_qty,
       tsfhead.from_loc_type,
       tsfhead.from_loc
FROM tsfalloc,
     tsfhead,
     tsfdetail,
     v_restart_tsfalloc vrt
WHERE tsfalloc.tsfalloc_no      = tsfhead.tsfalloc_no
   AND tsfhead.tsf_no          = tsfdetail.tsf_no
   AND tsfhead.status          = 'A'
   AND tsfalloc.reserve_date    is not NULL
   AND tsfalloc.reserve_date    <=
       TO_DATE(:ov_tomorrow,'YYYYMMDD')
   AND tsfdetail.allocated_qty  > NVL(tsfdetail.fill_qty, 0)
   AND nvl(tsfdetail.tsf_qty,0) = 0
   AND vrt.driver_value         = tsfalloc.tsfalloc_no
   AND vrt.num_threads          = :ora_restart_number_threads
   AND vrt.thread_val           = :ora_restart_thread_val
   AND (tsfalloc.tsfalloc_no    > NVL(:ora_restart_tsfalloc, -999)
   OR (tsfalloc.tsfalloc_no     = :ora_restart_tsfalloc
   AND tsfdetail.sku            > :ora_restart_sku))

```

```
GROUP BY tsfalloc.alloc_method,  
         tsfalloc.tsfalloc_no,  
         tsfdetail.sku,  
         tsfhead.from_loc,  
         tsfhead.from_loc_type  
ORDER BY tsfalloc.tsfalloc_no,  
         tsfdetail.sku;
```

Shared Modules

ITEMLOC_QUANTITY_SQL.GET_WH_CURRENT_AVAIL – gets the stock available for reservation at the warehouse (= stock on hand – reserved on other transfers - RTV - unavailable))

TSF_ATTRIB_SQL.GET_PACKSIZE – gets the pack size based on the from warehouse and the destination location

ROUNDING_SQL.QTY_TO_PACKSIZE – rounds the fill quantity to pack size

Function Level Description

init()

Initialize restart recovery – call restart_init

Select vdate from period

process()

open driving cursor

loop

fetch cursor into variables

if (not found) break

get_avail_stock()

determine_stock_ratio()

loop through tsfdetail

fetch into variables

if (not found) break

call TSF_ATTRIB_SQL.GET_PACKSIZE

call ROUNDING_SQL.QTY_TO_PACKSIZE

(round to pack size)

```
        update tsfdetail – set fill qty and transfer qty = (allocated qty –  
        fill qty) * ratio
```

```
    end loop
```

```
end loop
```

```
close driving cursor
```

```
get_avail_stock()
```

- gets the current available stock for reservation for the item at the source warehouse

```
call ITEMLOC_QUANTITY_SQL.GET_WH_CURRENT_AVAIL
```

```
determine_stock_ratio()
```

- Determine the ratio of stock each to location will receive. If the allocation method = 'A' - allocation quantity - then whichever location requested the most stock receives all of it first. Else if the allocation method is 'P' - pro-rate (or 'C' - custom) then the stock will be distributed based on the ratio of available to requested.

```
if (alloc_method == 'A')
```

```
    ratio = 1
```

```
else (alloc_method is 'P' or 'C')
```

```
    ratio = (available quantity/sum of allocated quantity)
```

```
    if (ratio > 1) ratio = 1
```

end if

final()

call restart_close()

Technical Issues

N/A

[txrposdn.pc]

Design Overview

This batch program will process records off of the geocode_txcde, product_taxcode, and tax_rates tables, and write all item/site combinations to the TIF_EXPLODE table for tax rate processing to be sent to the POS flat file. (created by tifposdn.pc) This program will write all item/category/store combinations that are affected by changes to these three tables by pulling the records from the tables where the download indicator is 'Y' and the start date less than or equal to (current date +2), where the end date is equal to (current date +2), or where the download indicator is 'Y' and the end date is equal to current date or (current date +1). After each record is processed, the download_ind flag will be set to 'N'.

Scheduling Constraints

Processing Cycle:	Phase 4 (daily)
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	Run tifposdn.pc to create a flat file.
Threading Scheme:	N/A

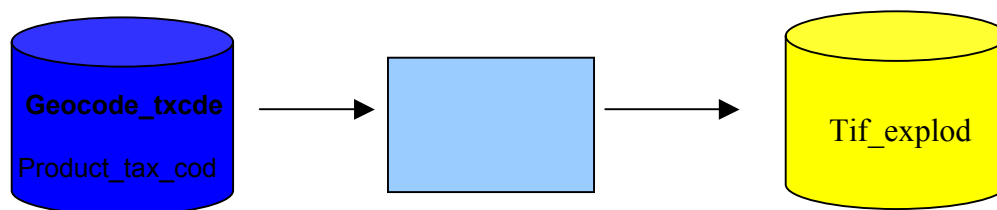
Restart Recovery

Logical Unit of Work = item/store combination

Driving Cursor:

(See cursors defined in process function)

Program Flow



Shared Modules

None

Function Level Description

Init()

Get system date.

This function initializes restart/recovery for this program. It also retrieves system variables and calls a function to size the arrays used in this program.

Process()

This function drives the processing of the program. This function will insert all item/store combinations that are affected by a change on the geocode_txcode, product_tax_code, or tax_rates tables to the TIF_EXPLODE table. After a change on the geocode_txcode, product_tax_code, or tax_rates table has been processed, the download_ind will be set to 'N' so the change is not sent to the POS system more than once for normal processing.

Technical Details

Processing in the function is broken up into three sets of item/store combinations: those found on the WIN_SKUS table, those found on the RAG_SKUS_ST table, and those on the PACKSTORE table.

A separate function will process the category/store combinations that need to be downloaded.

A function is defined for each of these three cases. In each function data is fetched from the table and placed into arrays. The data in each element is then processed and collected for an array insert into the tif_explode table.

The function should fetch records from the driving cursors into arrays. The arrays should be sized to match the value of the maximum commit counter on the restart_control table. Once the records are fetched, each record should be processed in a for-loop. After all of the records have been processed in the for loop, the restart_commit logic should be called. Transaction counters, etc. should be saved into the application image array string that is passed to the restart_control function.

Cursors used in this function:

C_WIN_STORE_CHANGE

This cursor will retrieve item/store data from the win_store table that meet the following criteria:

The store is listed on the geocode_store table with the following criteria:

- start_date <= (current date + 2) and download_ind = 'Y'
- end_date = (current date +2) and download_ind = 'N'
- (end_date = current date or end_date = (current_date +1)) and download_ind = 'Y'

In this case, all items are retrieved that will need a new tax calculation performed. These item/store combinations will be written to the tif_explode table with the current_rate_ind and changed_ind defined as described below.

An item on the wins_stores table is listed on the product_tax_code table with the following criteria:

- start_date <= (current date + 2) and download_ind = 'Y'
- end_date = (current date +2) and download_ind = 'N'
- (end_date = current date or end_date = (current_date +1)) and download_ind = 'Y'

The tax rate for the item (as defined by the tax_type_id and tax_jurisdiction_id on the tax_rates and product_tax_code tables) meet the following criteria:

- start_date <= (current date + 2) and download_ind = 'Y'
- end_date = (current date +2) and download_ind = 'N'
- (end_date = current date or end_date = (current_date +1)) and download_ind = 'Y'

In all three cases, the values of current_rate_ind and changed_ind are defined as such:

If the start_date or end_date for the record processed is equal to current day or (current day + 1) and the download_ind = 'Y' then set L_current_ind = 'N' and L_changed_ind = 'Y'. Otherwise set L_current_ind = 'N' and L_changed_ind = 'N'.

C_RAG_SKUS_ST_CHANGE

The functionality of this cursor is identical to the C_WIN_STORES_CHANGE cursor. In this case, however, item/store combinations are retrieved from the rag_skus_st table.

C_TAX_RATES_CHANGE

Again, the functionality is identical to C_WIN_STORES_CHANGE. In this case the SKU value is replaced by PACK_NO, and retrieved from the packstore table.

C_CATEGORY_CHANGE

This cursor will retrieve category/store data from the geocode_store table that meet the following criteria:

The store is listed on the geocode_store table with the following criteria:

- start_date <= (current date + 2) and download_ind = 'Y'
- end_date = (current date +2) and download_ind = 'N'
- (end_date = current date or end_date = (current_date +1)) and download_ind = 'Y'

In this case, all categories are retrieved that will need a new tax calculation performed. These category/store combinations will be written to the tif_explode table with the current_rate_ind and changed_ind defined as described below.

The category is listed on the product_tax_code table with the following criteria:

- start_date <= (current date + 2) and download_ind = 'Y'
- end_date = (current date +2) and download_ind = 'N'
- (end_date = current date or end_date = (current_date +1)) and download_ind = 'Y'

The tax rate for the category(as defined by the tax_type_id and tax_jurisdiction_id on the tax_rates and product_tax_code tables) meet the following criteria:

- start_date <= (current date + 2) and download_ind = 'Y'
- end_date = (current date +2) and download_ind = 'N'
- (end_date = current date or end_date = (current_date +1)) and download_ind = 'Y'

In all three cases, the values of current_rate_ind and changed_ind are defined as such:

If the start_date or end_date for the record processed is equal to current day or (current day + 1) and the download_ind = 'Y' then set L_current_ind = 'N' and L_changed_ind = 'Y'. Otherwise set L_current_ind = 'N' and L_changed_ind = 'N'.

Functional Overview

The functions contained within the program are as follows:

`int main()`

This function performs initial logon to the ORACLE database, and drives the program through initial, process, and final stages. All writing to the log file occurs here as well.

`int init()`

This function provides initial processing. Initial restart/recovery functions are provided here. Calls to `size_arrays()` and `get_vdate()` are also included.

`int size_arrays()`

This function retrieves memory from the memory heap, and allocates them to arrays used for data processing using the C `calloc` function. The size of these arrays is determined by the value of `commit_max_counter` on the `restart_control` table.

`int get_vdate()`

Retrieves the vdate from the period table in YYYYMMDD format.

`int process()`

Drives data processing by calls to `fetch_win_store_data`, `fetch_rag_skus_st_data` and `fetch_packstore_data`. The function also updates the `download_ind` on all associated tables (`geocode_store`, `product_tax_code`, `tax_rates`) to a value of 'N'.

`int set_changed_ind(int ii_dstart_date, int ii_dend_date, int ii_dnld_date)`

This function returns the value of `changed_ind` based on the criteria listed under the description for the cursor `C_WIN_STORES_CHANGE` written above.

`int fetch_win_stores_change()`

`int fetch_rag_skus_st_change()`

`int fetch_packstore_change()`

These functions provide nearly identical functionality. The appropriate cursor (as defined above) is used to retrieve item/store combinations, as well as the date information and download indicator. This information is used to set the values of `changed_ind` and `current_rate_ind`. All values are then placed in an array to be inserted into the `tif_explode` table. The functions `fill_insert_array()` and `insert_items()` are called to handle input into the database.

`int fetch_category_change()`

The appropriate cursor (as defined above) is used to retrieve category/store combinations, as well as the date information and download indicator. This information is used to set the values of `changed_ind` and `current_rate_ind`. All values are then placed in an array to be inserted into the `tif_explode` table. The functions `fill_insert_array()` and `insert_items()` are called to handle input into the database.

```
int fill_insert_array(char *is_store, char *is_sku, int li_current_rate_ind, int  
ii_changed_ind)
```

This function uses the values passed in to create a new record to be inserted into the database. These records are first checked against the database, and the data previously entered into the array to insure that duplicate records are not inserted into the database.

```
int insert_items()
```

This function inserts the previously constructed input array into the database.

```
int final()
```

Provides final processing. All previously allocated memory is freed, and final restart/recovery processing is performed.

I/O Specification

N/A

[Txrtupld.pc]

Design Overview

This program will be used to upload tax rates from an outside source into the tables into the RMS. The upload will provide the means to create new tax jurisdictions, tax codes, and tax rates. In addition to the ability to insert new tax codes and rates it will also be able to update the rates for existing tax codes.

Tax rates contain a rate, start date and end date(not required). Tax rates are associated to tax codes. A tax code can have many rates associated to it, however not more than one rate can be effective at one time. Thus when new rates are uploaded into the system for existing records, validation will occur to ensure that the start and end dates of the tax rates are not overlapping. If an overlap is found then the record will be written out to a reject file where it can be investigated further.

The input file will contain the tax jurisdiction, tax type, level and tax rates which will allow the batch programs to correctly update the tax rates to reflect any changes that have occurred.

Assumption: The tax codes that are in the input file will be the same tax codes in the Retek system. This program will not be responsible for matching, mapping, and/or converting the data from an outside source to the data that is present in the Retek system. Delete functionality will be carried out using the functionality of the end dates on the tax rates. This program will not physically delete any information off of the tables.

Scheduling Constraints

Processing Cycle: ad hoc

Program Flow

Main()

This function provided the structure for the program and outlines the flow of the program depending on when what function is being called.

Init()

Process()

Final()

Init()

Initialize the restart variables that will be used from restart recovery through out the program.

Open a temporary file that will be used to create the reject file when the program is completed.

Process()

In the beginning of this function create a loop to obtain all of the THEAD records that exist on the input file. Once the FTAIL is reached then exit out of the loop and continue on with the remaining processing.

Call the function validate_THEAD() this function will store all of the information in the THEAD line into local variables.

Call the function tax_jurisdiction ()

Call the function tax_codes()

Call the function tax_rates()

Validate_record()-

This function will check to ensure that the dates entered are valid dates on the calendar. Additional validation will occur to ensure that the date passed in are greater than or equal to today's date. If an invalid date is passed in , the record will be written out the reject file for further investigation.

tax_jurisdiction()-

This function checks to see if the current jurisdiction being processed exists currently on the tax_jurisdictions table.

If the tax_jurisdiction already exists in the system and does not need to be added.

If the tax_jurisdiction does not exist in the system ,it is a new tax_jurisdiction and it needs to be added. This will be done by filling the insert_tax_jurisdictions array.

tax_codes()-

This function checks to see if the current tax code being processed exists currently on the tax_codes table.

If the tax code already exists in the system and does not need to be added.

If the tax code does not exist in the system, it is a new tax code and it needs to be added into the system. This will be done by filling the insert_tax_codes array.

tax_rates()-

This function checks to see if the current tax rate being process exists currently on the tax_rates table.

If it does exist then the tax rate already exists in the system and does not need to be added.

Call the function update_tax_rates

If it does not exist in the system it is a new tax rate and it needs to be added into the system. This will be done by filling the insert_tax_rates array. This is done by calling the function fill_tax_rates_array

Update_tax_rates()

This function should check to see if there currently is an ongoing tax rate in the system.

If there is an ongoing rate then the first rate must be ended the day before the new rate is to start.

Validate that the current tax rate's date ranges do not conflict with the other date ranges that are currently set up for the current tax code. If conflicts are found then the record will be written out to the reject file for further investigation.

Call the function fill_tax_rates_array

Process_record()

This function will insert the arrays the were created in the functions above. The arrays should be inserted into the tables in the following order:

Insert_tax_jurisdiction

Insert_tax_code

Insert_tax_rate

Final()

Close the reject file and exit out of the program successfully.

I/O Specification

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	File head descriptor	Char(5)	FHEAD	Describes file line type
	Line id	Char(10)	0000000001	Sequential file line number
	Gentran ID	Char(5)	'TXUP'	Identifies which translation Gentran uses
	Current date	Char(14)		File date in YYYYMMDDHH24MISS format
FDETL	File record descriptor	Char(5)	THEAD	Start of an invoice transaction
	Line id	Char(10)		Sequential file line number
	Tax Jurisdiction	Number(8)		Tax Jurisdiction
	Tax Jurisdiction Description	Char(40)		Tax Jurisdiction description
	Tax Level	Char(6)		Tax level of the Jurisdiction
	Tax Type	Char(6)		Tax Type
	Tax Rate	Number(8,5)		Tax rate of the tax code
	Start Date	Char (14)		Start Date of the tax rate
	End Date	Char (14)		End Date of the tax rate
FTAIL	File record descriptor	Char(5)	FTAIL	Marks end of file
	Line id	Char(10)		Sequential file line number
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

Mapping of the fields:

Comets	Retek
Tax jurisdiction	Tax Jurisdiction
Geocode Description	Tax Jurisdiction Description
Level	Tax Level
3 rd digit of the Tax item field	Tax Type

VAT Rate Maintenance [vatdlxpl]

Design Overview

Value Added Tax (VAT) is a tax imposed by some governments on goods that have realized an increase in value.

As with price zones and cost zones, individual stores belong to a particular VAT region. VAT regions are areas that contain VAT code and VAT rate information that applies to stores in that region and are based on VAT rates defined by the government. VAT codes can be set up by Retek users to identify a particular VAT rate percentage. Only one VAT rate can be active for any one VAT code.

VAT rates are stored at the SKU level in Retek on the VAT_SKU table. On this table, records with a past active date serve as an audit trail to track what VAT code and VAT rate a particular SKU has had or currently has. The VAT_SKU record with the most recent active date holds the SKU's current VAT code. The VAT rate related to a particular VAT code is found on the VAT_CODE_RATES table. Records on VAT_SKU with a future active date functionally represent a pending change in the VAT code for that particular SKU, effective on that date and reflecting the corresponding VAT rate associated with that VAT code on the VAT_CODE_RATES table.

Records on the VAT_CODE_RATES table functionally represent a change in the VAT rate for a particular VAT code. When a record exists on this table with an active date of tomorrow, this program updates the VAT rate on any future VAT_SKU records associated with that VAT code to the new VAT rate defined in the VAT_CODE_RATES table. Also, if the latest VAT_SKU record for any SKU contains the VAT code that is changing, a new VAT_SKU record is inserted for that SKU with an active date of tomorrow and the new VAT rate associated with the VAT code.

Tables Affected:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
VAT_CODE_RATES	Yes	Yes	No	No	No
VAT_SKU	Yes	Yes	Yes	Yes	No

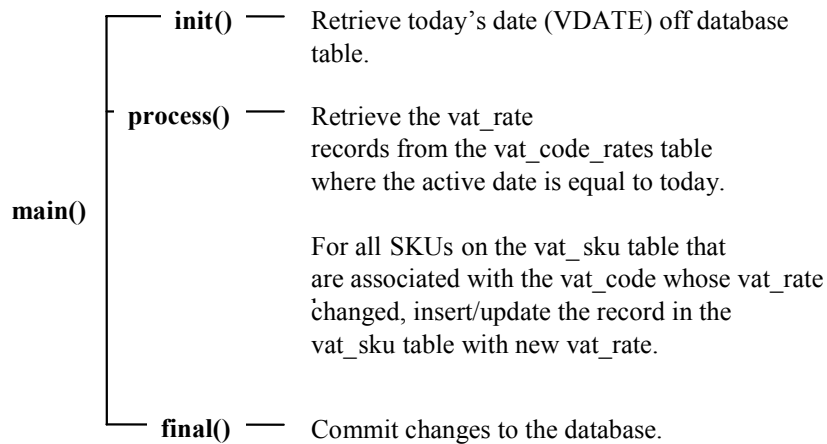
Scheduling Constraints

Processing Cycle:	Phase I
Scheduling Diagram:	N/A
Pre-Processing:	N/A
Post-Processing:	N/A
Threading Scheme:	N/A

Restart Recovery

The logical unit of work will be based on VAT code. Restart/recovery will be based on VAT code. It is recommended that records be committed after a set of 10,000 rows has been processed.

Program Flow



Shared Modules

N/A

Function Level Description

init():

Retrieve tomorrow's date off the period table.

process():

This function will retrieve all records off of the VAT_CODE_RATES table with an active date of tomorrow. If any records exist on the VAT_SKU table with an active date on or after tomorrow for the VAT code fetched off of the VAT_CODE_RATES table, their VAT rate is updated with the VAT rate fetched off of the VAT_CODE_RATES table. Also, if a SKU is currently assigned the VAT code with the changing VAT rate, a new record is inserted into VAT_SKU for the new VAT code/VAT rate combination with an active date of tomorrow.

update_vat_sku():

This function updates any future VAT_SKU records with the new VAT rate fetched off the VAT_CODE_RATES table. It utilizes array processing to handle the update.

insert_vat_sku():

Inserts a new record into the VAT_SKU table for any SKUs that are currently associated with the VAT code fetched from the VAT_CODE_RATES table. A new record is inserted into VAT_SKU for the new VAT code/VAT rate combination with an active date of tomorrow.

Technical Issues

This program implements the use of array processing to handle the inserts/updates of the VAT_SKU table.

Vendor Replenished Order Build [vrplbld]

Design Overview

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
EDI_ORD_TEMP	No	Yes	No	No	No
ORDHEAD	Yes	Yes	Yes	No	No
ORDSKU	No	No	Yes	Yes	No
ORDLOC	No	No	Yes	No	No
PERIOD	Yes	Yes	No	No	No
WIN_STORE	Yes	Yes	No	No	No
WIN_WH	Yes	Yes	No	No	No
RAG_SKUS_ST	Yes	Yes	No	No	No
RAG_SKUS_WH	Yes	Yes	No	No	No
UNIT_OPTIONS	No	Yes	No	No	No
ITEM_SUPP_COUNTRY	Yes	Yes	No	No	No
SUPP_IMPORT_ATTR	Yes	Yes	No	No	No
SUPS	Yes	Yes	No	No	No
ORDSKU_HTS	Yes	Yes	Yes	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
DEAL_CALC_QUEUE	No	Yes	Yes	Yes	No
ORD_INV_MGMT	No	No	Yes	No	No

Note: ORDHEAD (select via index) is referenced via the function ORDER_NUMBER_SQL.NEXT_ORDER_NUMBER.

Indexes

- ORDHEAD(order_no),
- ORDSKU(order_no,sku),
- WIN_STORE(sku,store),
- WIN_WH(sku,wh),
- RAG_SKUS_ST(sku,store),
- RAG_SKUS_ST(sku,wh)

Function: To continue the process started by the program ediupack of building RETEK orders that reflect the vendor-generated orders as received through the EDI 855.

The program ediupack.pc processes the EDI 855's that have been received from vendors. Rows have been inserted onto the table EDI_ORD_TEMP for use by this program. The EDI_ORD_TEMP table contains all items (skus) which were included on the EDI 855, along with the vendor order number with which they were originally associated.

The items are then consolidated so that one RETEK order is created for each vendor-order-number/department/supplier or vendor-order-number/supplier or depending on UNIT_OPTIONS.dept_level_orders (the latter occurs when UNIT_OPTIONS.dept_level_orders = 'N').

Some details for each sku are retrieved from the item/location tables and ITEM_SUPPLIER_COUNTRY and stored in ORDSKU and ORDLOC so that subsequent changes to the sku do not affect the outstanding order.

A_order_amt is updated on the OTB table. If the supplier is foreign and OTB is calculated at cost, the added amount is total_duty + total_cost. If the supplier is not foreign and OTB is calculated at cost, the added amount is total_cost. Finally, if OTB is calculated at retail, the added amount is total_retail.

In records written to the ORDHEAD table:

- FREIGHT_TERMS is given the value fetched from SUPS.FREIGHT_TERMS
- ORIG_APPROVAL_DATE is assigned the value of the PERIOD.VDATE
- ORIG_APPROVAL_ID is assigned the value 'EDI855'

If an item is ordered for a location at which it does not have an item/location record, a record is built by calling the stored procedure NEW_ITEM_LOC.

The order numbers are allocated automatically on a 'next available' basis.

Note: The vendor minimum order quantity is not considered when writing to the order tables because the quantity is determined by vendor. Thus, this quantity is not stored in the edi_ord_temp table.

Re-run: If this program aborts and terminates normally or terminates abnormally restart after setting restart_flag = 'Y' on restart_program_status table for current restart_name, schema, & thread_val.

Scheduling Constraints

Processing Cycle: PHASE 3

Scheduling Diagram: N/A

Pre-Processing: N/A

Post-Processing: Vrplbld_post in prepost.pc truncates edi_ord_temp table

Threading Scheme: SUPPLIER

Restart Recovery

The logical unit of work (LUW) for vrplbld.pc will be a count of records pulled from the edi_temp_ord table. The number of records will be determined by the commit_max_ctr field on the restart_control table. Further, one or more of the key values (vendor_order_no, supplier, dept) must change before a commit event can take place. This will ensure that only whole order will be saved, i.e. that a commit will never take place before all of the order detail records have been processed.

The commit_max_ctr field should be set to prevent excessive rollback space usage. Given the use of multiple threading, the recommended commit counter setting is 5000 records (subject to change based on experimentation).

Two cursors are defined, but only one cursor is opened for fetching (the criteria is based on whether or not departmental level ordering is allowed.)

```
EXEC SQL DECLARE c_skus_d CURSOR FOR
```

```
    SELECT isc.origin_country_id,
```

```
           isc.supp_pack_size,
```

```
           sia.agent,
```

```
           sia.lading_port,
```

```
           sia.discharge_port,
```

```
           s.currency_code,
```



```

        e.vendor_order_no,
        e.supplier,
        e.dept,
        e.wh_or_store_c,
        e.wh_or_store,
        e.sku,
        e.sku_desc,
        e.upc,
        e.upc_supplement,
        e.unit_cost,
        e.qty_ordered,
        TO_CHAR(e.written_date, 'YYYYMMDD'),
        TO_CHAR(e.not_before_date, 'YYYYMMDD'),
        TO_CHAR(e.not_after_date, 'YYYYMMDD'),
        TO_CHAR(e.not_after_date, 'DD'),
        TO_CHAR(e.not_after_date, 'MM'),
        TO_CHAR(e.not_after_date, 'YYYY'),
        ';' || e.vendor_order_no ||
        ';' || TO_CHAR(e.dept) ||
        ';' || TO_CHAR(e.supplier)
FROM item_supp_country isc,
     sup_import_attr sia,
     sups s,
     edi_ord_temp e,
     v_restart_supplier rv
WHERE isc.item          = e.sku

```

```
AND isc.primary_country_ind = 'Y'
AND isc.supplier          = e.supplier
AND sia.supplier (+)      = e.supplier
AND s.supplier            = e.supplier
AND rv.driver_value       = e.supplier
AND rv.driver_name        = :os_restart_driver_name
AND rv.num_threads        = :oi_restart_num_threads
AND rv.thread_val         = :oi_restart_thread_val
AND (e.vendor_order_no > NVL(:os_restart_vndr_ord_no,-999) OR
      (e.vendor_order_no = :os_restart_vndr_ord_no AND
        (e.dept > :os_restart_dept OR
          (e.dept = :os_restart_dept AND
            e.supplier > :os_restart_supplier)
        )
      )
    )
ORDER BY e.vendor_order_no,
         e.dept,
         e.supplier,
         e.wh_or_store_c,
         e.wh_or_store,
         e.sku;
```

```
EXEC SQL DECLARE c_skus_s CURSOR FOR
SELECT isc.origin_country_id,
```

```
isc.suppack_size,
sia.agent,
sia.lading_port,
sia.discharge_port,
s.currency_code,
e.vendor_order_no,
e.supplier,
e.dept,
e.wh_or_store_c,
e.wh_or_store,
e.sku,
e.sku_desc,
e.upc,
e.upc_supplement,
e.unit_cost,
e.qty_ordered,
TO_CHAR(e.written_date, 'YYYYMMDD'),
TO_CHAR(e.not_before_date, 'YYYYMMDD'),
TO_CHAR(e.not_after_date, 'YYYYMMDD'),
TO_CHAR(e.not_after_date, 'DD'),
TO_CHAR(e.not_after_date, 'MM'),
TO_CHAR(e.not_after_date, 'YYYY'),
';' || e.vendor_order_no ||
';' || TO_CHAR(e.dept) ||
';' || TO_CHAR(e.supplier)
FROM item_supp_country isc,
```

```
sup_import_attr sia,
sup_s s,
edi_ord_temp e,
v_restart_supplier rv
WHERE isc.item          = e.sku
AND isc.primary_country_ind = 'Y'
AND isc.supplier        = e.supplier
AND sia.supplier (+)     = e.supplier
AND s.supplier           = e.supplier
AND rv.driver_value     = e.supplier
AND rv.driver_name      = :os_restart_driver_name
AND rv.num_threads      = :oi_restart_num_threads
AND rv.thread_val       = :oi_restart_thread_val
AND (e.vendor_order_no > NVL(:os_restart_vndr_ord_no,-999) OR
      (e.vendor_order_no = :os_restart_vndr_ord_no AND
        e.supplier > :os_restart_supplier)
      )
ORDER BY e.vendor_order_no,
         e.supplier,
         e.wh_or_store_c,
         e.wh_or_store,
         e.sku;
```

Program Flow

N/A

Shared Modules

NEW_ITEM_LOC
ORDER_NUMBER_SQL.GET_NEXT_ORDER_NUMBER
CURRENCY_SQL.GET_RATE
CAL_TO_454_LDOW
ORDER_SETUP_SQL.DEFAULT_ORDHEAD_DOCS
ORDER_SETUP_SQL.DEFAULT_ORDSKU_DOCS
ORDBA_CALC_SQL.ORDER_COST
ORDER_EXPENSE_SQL.INSERT_COST_COMP
ORDER HTS_SQL.DEFAULT_CALC HTS
OTB_SQL.ORD_APPROVE

Function Level Description

init()
select department level orders flag from unit options
select base country, latest ship days, FOB information, ELC indicator, EDI cost
override indicator, calendar 454 indicator, and bill-to location from system
options
select vdate from period
call restart initialization logic
process()
call open_cursor to open appropriate cursor (department level or multi-
department orders)

call fetch_cursor for prime fetch of driving cursor

while

if supplier, department or vendor order number changed then

call create_header

call header_defaults

if sku changed

call create_sku

call add_loc

call fetch_cursor

if supplier, department or vendor order number changed then

if the ELC indicator is Y, call add_cost_comp

call sku_defaults

call apply_disc

call update_header

else if the sku changed

if the ELC indicator is Y, call add_cost_comp

call sku_defaults

call restart commit logic

end while

call sku_defaults

call apply_disc

call update_header

open _cursor()

open appropriate cursor depending on what type of orders are allowed.

fetch _cursor()

fetch records from appropriate cursor

item_active ()

returns 1 if item is active (0 if inactive) on win_store, win_wh, rag_skus_st or

rag_skus_wh

create_header()

call function to get the next order number

(ORDER_NUMBER_SQL.NEXT_ORDER_NUMBER)

call CAL_TO_454_LDOW procedure to convert order's not_after_date to the end-of-week date to update otb_eow_date field

call CURRENCY_SQL.GET_RATE to get the exchange rate for the supplier's currency.

insert record into ordhead

header_defaults()

Call ORDER_SETUP_SQL.DEFAULT_ORDHEAD_DOCS to default the necessary documents for the order header. Also call ORDER_SETUP_SQL.

DEFAULT_ORDER_INV_MGMT to appropriate supplier inventory management information to the order with NULL inserted for the scaling constraints, since a vendor generated order is not eligible for scaling.

update_header()

Check to see if any items on the order come from a country other than the ordhead import country. If so, update the import_order_ind to 'Y' on ordhead. If this is an import order and the ELC indicator is 'Y', call check_hts to make sure approved HTS codes exist for all items.

sku_defaults()

Call ORDER_SETUP_SQL.DEFAULT_ORDSKU_DOCS to default necessary documents for items in order.

apply_disc()

if edi_cost_override_ind = 'Y' then insert a record into the DEAL_CALC_QUEUE table if the current order number is not in the table, or update the order_appr_ind field to 'Y' if the order number is already in the table.

call update_OTB

add_cost_comp()

Call ORDER_EXPENSE_SQL.INSERT_COST_COMP,

ORDER HTS_SQL.DEFAULT_CALC HTS and

ELC_CALC_SQL.CALC_COMP.

create_sku ()

if edi_cost_override_ind = 'Y' insert record into ordsku with the cost_source = 'MANL'. If edi_cost_override_ind != 'Y', insert a record into ordsku with the cost_source = 'NORM';

update_OTB ()

call check_negative_cost

if check_negative_cost returns 1 then update ordhead.status to 'W' and reject code to 'NC'

else, call OTB_SQL.ORD_APPROVE to update OTB

add_loc ()

insert into ordloc

check_negative_cost()

check for a record on the ordsku table with unit_cost < 0

if a record is found, return 1

else, return 0

check_hts()

For a given order number, check if items on the order all have approved HTS codes, by checking the status on the ordsku_hts table. There should be the same number of skus for the order on both ordksu and ordsku_hts (otherwise the order immediately fails the check) and the statuses must all be 'A'.

set_worksheet_status()

For a given order number and reject code, update the status on the ordhead table to 'W'orksheets and update the reject code to whatever was passed in.

order_exists()

For a given order number, check to see whether or not the order number is on the DEAL_CALC_QUEUE table.

final()

Call restart/recovery close logic.

I/O Specification

N/A

Technical Issues

N/A

Wastage Adjustment [wasteadj]

Design Overview

This program will reduce inventory of spoilage type wastage items to account for natural wastage that occurs over the shelf life of the product. Only items with spoilage type wastage will be affected by this program. Sales type wastage will be accounted for at the time of sale.

Spoilage type wastage is due to the natural loss of a product over its shelf life. For example, as sausage sits on the shelf it loses a certain percentage of its weight each day due to evaporation of the water that is in the sausage. Therefore, a sausage that weighs one pound today may only weight .9 pounds tomorrow due to the loss of water from the sausage. This new batch program will reduce the physical on hand inventory for a store based on the daily wastage percentage which will be entered by the retailer at the item location level:

$$\text{New Stock on Hand} = \text{Actual Stock on Hand} * (1 - \text{Daily Wastage \%})$$

When stock is reduced, Merchandising will process these records just as any other stock adjustment.

By automatically reducing stock on hand based on the wastage percentage and the shelf life of an item, the on hand inventory in Merchandising will be more accurate than if these adjustments were not made.

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
DESC_LOOK	No	Yes	No	No	No
WIN_STORE	No	Yes	No	Yes	No
RAG_SKUS_ST	No	Yes	No	Yes	No
INV_ADJ	No	No	Yes	No	No
TRAN_DATA	No	No	Yes	No	No
PERIOD	No	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	Yes	No

Scheduling Constraints

Processing Cycle: Phase III (Before stock ledger processing)

Scheduling Diagram:

Pre-Processing: Run this program before the stock ledger roll up programs to make sure that the stock adjustments taken during the current day are credited to the appropriate day.

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

Restart recovery is based on last item processed. The program will commit only when the commit max counter is reached.

Program Flow

N/A

Shared Modules

N/A

Function Level Description

init

- Initialize restart variables.
- Allocate memory to arrays for array processing
- Collect system information

process

- Fetch SKUs with wastage type spoilage ('SP')
- Retrieve single SKU from fetch array for processing
- If a staple SKU:
 - retrieve appropriate information from win_store
 - add information to insert/update arrays

- If a fashion SKU, retrieve appropriate information from rag_skus_st
 - retrieve appropriate information from rag_skus_st
 - add information to insert/update arrays

create_inv_adj

- Adds a record to the inv_adj insert array

create_tran_data

- Adds a record to the tran_data insert array

update_stock

- Performs array updates of win_store and rag_skus_st records

write_arrays

- Performs array inserts to inv_adj and tran_data

final

- Closes restart recovery

I/O Specification

N/A

Technical Issues

N/A

Customization Issues

N/A

Warehouse Store Assignment [whstrasg]

Design Overview

The whstrasg program is used to update the default warehouse attribute on the Store table and the sourcing warehousing replenishment attribute on the Replenishment Item Location table based on the warehouse-store assignments on the Warehouse Store Assignment table. The program will run nightly and on the night before the assignment date for a warehouse-store assignment, it will update the relevant tables with the new warehouse-store relationship. If the warehouse is to be assigned to an item on the Replenishment Item Location and the item is not associated to the warehouse, a non-fatal error message should be output indicating that the item is not associated to the warehouse. When updating the sourcing warehouse on the Replenishment Item Location table, a system indicator will determine whether the sourcing warehouse will be updated for warehouse-store replenishment records, cross-docked replenishment records or both.

Affected Tables:

TABLE	INDEX	SELECT	INSERT	UPDATE	DELETE
WH_STORE_ASSIGN	Yes	Yes	No	Yes	No
STORE	Yes	No	No	Yes	No
REPL_ITEM_LOC	Yes	Yes	No	Yes	No
WIN_WH	Yes	Yes	No	No	No
RAG_SKUS_WH	Yes	Yes	No	No	No
SYSTEM_OPTIONS	No	Yes	No	No	No
PERIOD	No	Yes	No	No	No

Scheduling Constraints

Processing Cycle: PHASE 3 (daily)

Scheduling Diagram: After all replenishment batch programs since the assignments are meant to be effective the next day, not the current day.

Pre-Processing: N/A

Post-Processing: Clear off the WH_STORE_ASSIGN table
(whstrasg_post)

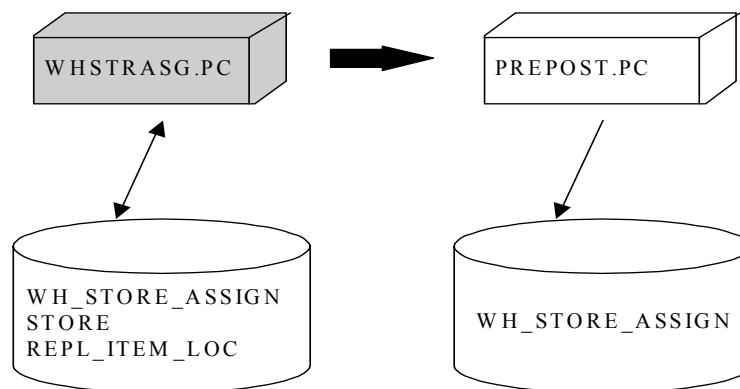
Threading Scheme: Store.

Restart Recovery

Restart/recovery for this program is set up at the store level.

The recommended commit counter setting is 2000 records (subject to change based on experimentation).

Program Flow



Shared Modules

N/A

Function Level Description

Init():

This function will call `retex_init()` to start up restart recovery and bring back the bookmark string in case of a restart. Fetches `vdate` from the `PERIOD` table; fetches `wh_store_assign_type` from the `SYSTEM_OPTIONS` table.

Process():

This function is the controller for most of the processing in the program. It opens and fetches from the driving cursor according to Retek standards, using array processing and handling restart/recovery

Once the records are fetched, call `update_repl_item_loc()` to fill the update array of the `REPL_ITEM_LOC` table then call `update_repl()` to update the `source_wh` field of the `REPL_ITEM_LOC` table.

For every for fetch loop, call `update_store()` to update the `default_wh` field of the `STORE` table and `processed_ind` of the `WH_STORE_ASSIGN` table.

`Final()`:

This function will call `retek_close()` to wrap up restart-recovery and populate the final error message, if necessary.

`Update_repl_item_loc()`:

This function will check each item depending on the `system_ind` type, check if `win_wh` or `rag_skus_wh` exists for the item/warehouse. If not, return a non-fatal error and write an error message to the error log. Since a pack item cannot be replenished, if the `system_ind` is 'P', also return a non-fatal error and write an error message to the error log. If there is no non-fatal error, fill the update array of the `REPL_ITEM_LOC` table.

`Init_arrays()`:

This function initializes the size of the array used for the driving cursor fetch the size of the restart max counter on `restart_control`.

`Update_repl()`:

This function will update the `source_wh` field of the `REPL_ITEM_LOC` table to the warehouse of the driving cursor using array update. This is by rowid.

`Update_store()`:

This function will update the `default_wh` field of the `STORE` table using array update, this is not by rowid. And update the `process_ind` field of the `WH_STORE_ASSIGN` table to 'Y' for all records that have been successfully processed, this is by rowid.

I/O Specification

N/A

Technical Issues

N/A

Chapter 17 - Overview of RMS Batch Programs

Batch Overview

This document describes the following features of Retek batch programs:

- Restart and recovery
- Multi-threading
- Commit thresholds
- Array processing
- Input and output formats to external applications and entities

Restart and Recovery

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

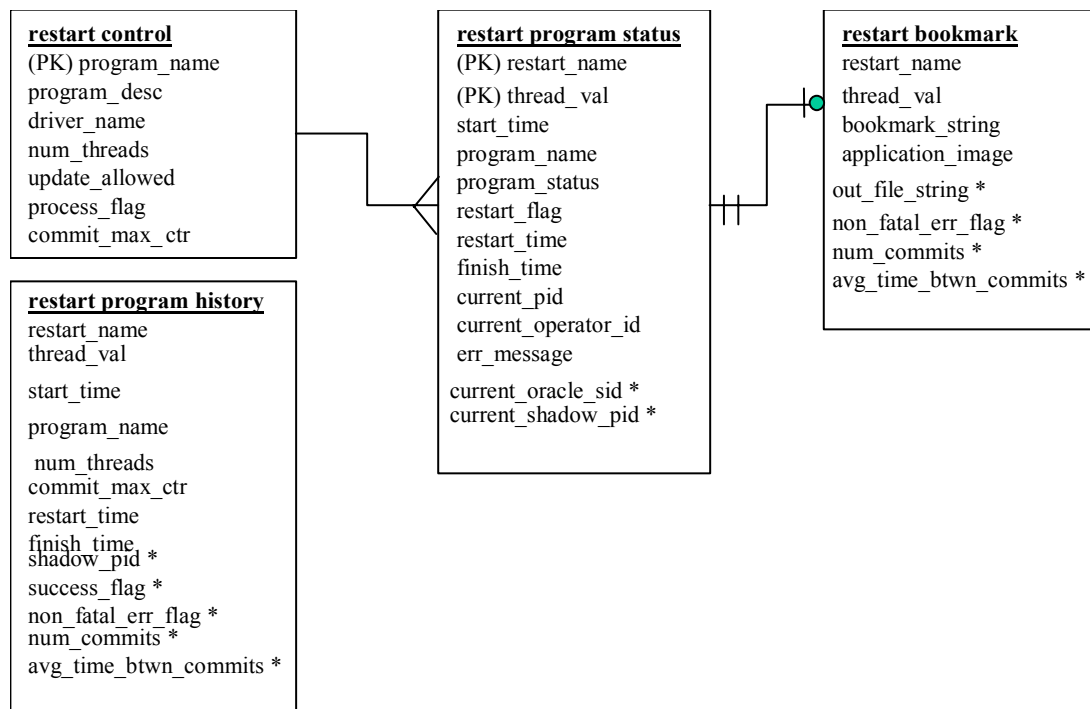
- recover a halted process from the point of failure
- prevent system halts due to large numbers of transactions
- allow multiple instances of a given process to be active at the same time

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

The restart capabilities will revolve around a program's logical unit of work (LUW). A batch program will process transactions and commit points will be enabled based on the LUW. LUWs will consist of a relatively unique transaction key (such as sku/store) and a maximum commit counter. Commit events will take place after the number of processed transaction keys meets or exceeds the maximum commit counter, e.g. every 10,000 sku/store combinations a commit will occur. 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 & definitions

The RMS restart/recovery process is driven by a set of four tables. Refer to Diagram 1 for the entity relationship diagram, followed by table descriptions.



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

Diagram 1 – Restart/recovery table relationships

restart_control

The restart_control table is the master table in the restart/recovery table set. One record will exist 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,
- and 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, e.g. department (non-updatable)
num_threads	Num	4	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 table is modified during restart/recovery initialization and close logic. For table-based, 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, 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	4	thread counter
start_time	date		dd-mon-yy hh:mi:ss
program_name	varchar2	25	program name
program_status	varchar2	15	started, aborted, 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

(PK) restart_name	varchar2	50	
(PK) thread_val	num	4	
(PK) start_time	date		
program_name	varchar2	25	
num_threads	num	4	
commit_max_ctr	num	6	
restart_time	date		
finish_time	date		
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, a record for it will exist 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

restart_name	varchar2	50	
thread_val	num	4	
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, e.g. 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 & 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, i.e. 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 and restart_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 the type of transaction being processed by the input file.

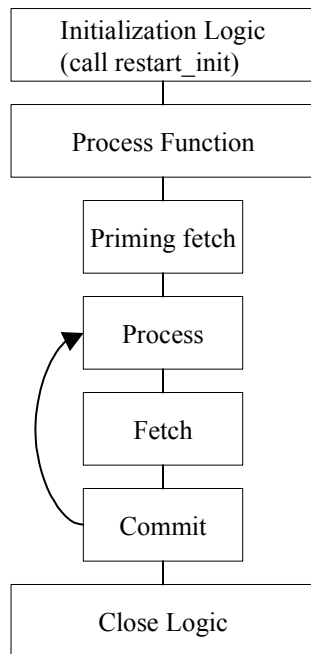


Diagram 2 - Table-Based Restart/Recovery Program Flow

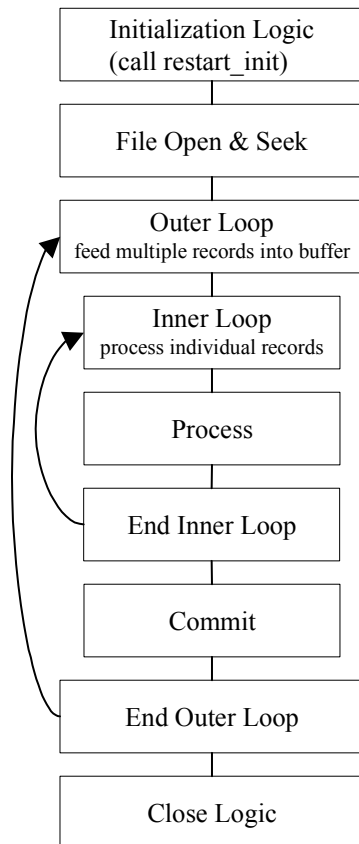


Diagram 3 - File-Based Restart/Recovery Program Flow

Initialization Logic:

- Variable declarations
- file initialization
- call restart_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()

Close logic:

- call restart_close()
- reset pointers
- close files/cursors

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.

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

and 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.

and 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)
- and the bookmark string of the last processed record is
- and increments the counter
- and 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 & 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'
- and the changes are committed.

parse_array_args:

This function parses a string into components and stuffs 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 & 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, e.g. counters & accumulators
- the name of the threading driver, e.g. 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

New Restart Headers & Libraries

The current RMS restart/recovery library has been modified 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

retk_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 `retk_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 `retk_init()` needs to populate are required to be passed in in a format known to `retk_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", "long", "string", or "rtk_file".

For type "int" or "long", use "" as sub_type.

For type "string", sub_type can only be "S" (start string) 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, Retek is currently maintaining two sets of restart/recovery libraries.

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

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

- 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 and long variables need to be passed by reference;
- Get all global and module level values from databases;
- Initialize records for RESTART_PROGRAM_STATUS and RESTART_BOOKMARK;
- Parse out user-specified initialization variables (variable arg list);
- Return NO_THREAD_AVAILABLE if no qualified record in RESTART_CONTROL or RESTART_PROGRAM_STATUS;
- Commit work.

int retek_commit(int num_args, ...)

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

- 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();
- Concatenate start_string either from passed in variables (table-based) or from fteell of input file pointers (file-based);
- Check if commit point reached (counter check and, if table-based, start string comparison);
- If reached, concatenated image_string from passed in variables (if needed) and call internal_commit() to get out_file_string and update RESTART_BOOKMARK;
- 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.

- 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;
- 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;
- For file-based, if pl_curren_count reaches pl_max_counter return 1; otherwise return 0.

int retek_force_commit(int num_args, ...)

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

- 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();

- Concatenate start_string either from passed in variables (table-based) or from full of input file pointers (file-based);
- Concatenated image_string from passed in variables (if needed) and call internal_commit() to get out_file_string and update RESTART_BOOKMARK;
- If table-based, increment pl_current_count and update ps_cur_string.

int retek_close(void)

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

- If gi_error_flag or NO_COMMIT command line option is TRUE, rollback all database changes;
- Update RESTART_PROGRAM_STATUS according to gi_error_flag;
- 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;
- If no gi_error_flag, delete RESTART_BOOKMARK record;
- Commit work;
- Close all opened file streams.

Int retek_refresh_thread(void)

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

- Updates the RESTART_PROGRAM_STATUS record for the current program's PROGRAM_STATUS to be 'ready for start';
- Delete any RESTART_BOOKMARK records for the current program;
- Commit 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.

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 (e.g. departments or stores) into particular threads given value (e.g. 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, however, 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.

Threading description

The use of multiple threads or processes in Retek 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 (e.g. 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. Clients 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`;
- and 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 client were to always use the same number of threads for a given driver (dept, store, etc.), then the view will be relatively small. As an example, if all of a client's programs threaded by department have a total of 5 threads, then the view will contain only one value for each department i.e. if 10 total departments, 10 rows will exist in `v_restart_dept`. However, if the client wants to have one of the programs to have ten threads, then there will exist 2 rows for every department - one for five total threads & one for ten total threads, i.e. if 10 total departments, 20 rows will exist in `v_restart_dept`. Obviously, clients should be advised 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 created the view for the multi-threadin 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 Sales Audit. Because Sales Audit is a 24X7 product, there is no batch window. Which means that there may be batch programs running at the same time that there are online users. Sales Audit 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 work load 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.

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 (i.e. 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:

<u>row #</u>	<u>restart_name</u>	<u>thread_val</u>	<u>program_name</u>	<u>etc...</u>
1	WinSal -main	1	WinSal	...
2	WinSal -main	2	WinSal	...

restart_program_status table after insert:

<u>row #</u>	<u>restart_name</u>	<u>thread_val</u>	<u>program_name</u>	<u>etc...</u>
1	WinSal -main	1	WinSal	...
2	WinSal -main	2	WinSal	...
3	WinSal -main	3	WinSal	...

restart_program_status table after delete

<u>row #</u>	<u>restart_name</u>	<u>thread_val</u>	<u>program_name</u>	<u>etc...</u>
1	WinSal -main	1	WinSal	...

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:

User 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 & 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/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.

Query-based Commit Thresholds

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. An LUW is comprised of a transaction key (such as sku/store) and a maximum commit counter. Commit events occur when 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.

Array Processing

Retek 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. Retek typically will not statically define the array sizes, but will use the restart maximum commit variable as a sizing multiple. Users should keep this in mind when defining the system's maximum commit counters.

Input / Output Formats

Retek 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 Retek to external systems is the main use of file based I/O.

General Interface Discussion

To simplify the interface requirements, Retek 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 Retek 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:

```
FHEAD0000000000STKU1996010100000019960929
```

```
FDETL0000000001SKU100000040000011011
```

```
FDETL0000000001SKU100000050003002001
```

```
FDETL0000000001SKU100000050003002001
```

```
FTAIL00000000020000000003
```

[javascript://Master/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

THEAD00000000020000000000123419960909120200000000003R

TDETL000000000300000000001234000001SKU10000012

TTAIL0000000004000001

THEAD000000000500000000001234199609091202001215720131R

TDETL000000000600000000001234000001UPC400100002667

TDETL000000000700000000001234000001UPC400100002643 0

TTAIL0000000008000002

FTAIL00000000090000000007

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

Record Name	Field Name	Field Type	Default Value	Description
	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
	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

Record Name	Field Name	Field Type	Default Value	Description
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

EDI

Starting with release 7.0, EDI files used or created by RMS are in a generic format. That is, 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 Retek 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 Retek client’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 Retek to continue to maintain code that supports any particular EDI standard. There are multiple viable standards that are utilized by vendors and clients. 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 Gentrans 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.

Interfaces Supported by RMS

The RMS batch modules include interfaces to many external systems. Point of sales systems both send sales files (via sales audit) and receive item pricing and maintenance files from RMS. Warehouse management systems and store management systems use RMS interface files to prepare transfers, and report transfers, receiving, inventory adjustments, returns to vendors, and stock take counts. EDI translation packages translate in-bound and out-bound transactions from standard EDI formats to Retek's genericized interface layouts. Planning and forecasting systems receive sales history and item information and provide order and replenishment information. Finally, external open to buy systems receive histories and provide new parameters to RMS.

Interface File Layouts

Specific file layout for the interface described above can be found in the Batch Designs section.

- Point of Sale: posupld, posdnld, promdnld
- Warehouse/store: tsfoupld, tsfiupld, tsfdnld, rcvupld, rtvupld, invaupld, stkupld
- EDI: ediupack, ediupadd, edidladd, ediupcat, edidlprd, edidlrd, ediupasn, edidlcnt, ediupavl
- Planning: plnu