

Oracle® Retail Price Management
Operations Guide Addendum
Release 11.0.12

November 2007

Copyright © 2007, Oracle. All rights reserved.

Primary Author: Nathan Young

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software—Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Value-Added Reseller (VAR) Language

- (i) the software component known as **ACUMATE** developed and licensed by Lucent Technologies Inc. of Murray Hill, New Jersey, to Oracle and imbedded in the Oracle Retail Predictive Application Server – Enterprise Engine, Oracle Retail Category Management, Oracle Retail Item Planning, Oracle Retail Merchandise Financial Planning, Oracle Retail Advanced Inventory Planning and Oracle Retail Demand Forecasting applications.
- (ii) the **MicroStrategy** Components developed and licensed by MicroStrategy Services Corporation (MicroStrategy) of McLean, Virginia to Oracle and imbedded in the MicroStrategy for Oracle Retail Data Warehouse and MicroStrategy for Oracle Retail Planning & Optimization applications.
- (iii) the **SeeBeyond** component developed and licensed by Sun Microsystems, Inc. (Sun) of Santa Clara, California, to Oracle and imbedded in the Oracle Retail Integration Bus application.
- (iv) the **Wavelink** component developed and licensed by Wavelink Corporation (Wavelink) of Kirkland, Washington, to Oracle and imbedded in Oracle Retail Store Inventory Management.
- (v) the software component known as **Crystal Enterprise Professional and/or Crystal Reports Professional** licensed by Business Objects Software Limited (“Business Objects”) and imbedded in Oracle Retail Store Inventory Management.
- (vi) the software component known as **Access Via™** licensed by Access Via of Seattle, Washington, and imbedded in Oracle Retail Signs and Oracle Retail Labels and Tags.
- (vii) the software component known as **Adobe Flex™** licensed by Adobe Systems Incorporated of San Jose, California, and imbedded in Oracle Retail Promotion Planning & Optimization application.
- (viii) the software component known as **Style Report™** developed and licensed by InetSoft Technology Corp. of Piscataway, New Jersey, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.
- (ix) the software component known as **i-net Crystal-Clear™** developed and licensed by I-NET Software Inc. of Berlin, Germany, to Oracle and imbedded in the Oracle Retail Central Office and Oracle Retail Back Office applications.
- (x) the software component known as **WebLogic™** developed and licensed by BEA Systems, Inc. of San Jose, California, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.
- (xi) the software component known as **DataBeacon™** developed and licensed by Cognos Incorporated of Ottawa, Ontario, Canada, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

Contents

Preface	vii
Audience	vii
Related Documents.....	vii
Customer Support.....	vii
Review Patch Documentation	viii
Oracle Retail Documentation on the Oracle Technology Network.....	viii
Conventions.....	viii
1 Introduction	1
2 Backend System Administration and Configuration	3
RPMTaskMDB.....	3
Tables Used by RPMTaskMDB.....	3
3 Technical Architecture	5
Asynchronous Processing.....	5
RPM-Related Java Terms and Standards.....	6
4 Batch Processes	7
Java Batch Processes	7
Java Batch Process Architectural Overview	7
Running a Java-Based Batch Process.....	7
Script Catalog	8
Scheduler and the Command Line.....	9
Functional Descriptions and Dependencies.....	9
Batch Process Scheduling	10
Threading and the RPM_BATCH_CONTROL Table	10
Return Value Batch Standards	10
Return Values.....	10
Batch Logging.....	11
Conflict Checking	11
InjectorPriceEventBatch Batch Design	19
LocationMoveBatch Batch Design.....	23
MerchExtractKickOffBatch Batch Design.....	24
NewItemLocBatch Batch Design	27
PriceChangeAreaDifferentialBatch Batch Design	29
purgeBulkConflictCheckArtifacts Batch Design	29
statusPageCommandLineApplication Batch Design.....	30
WorksheetAutoApproveBatch Batch Design.....	33

Preface

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

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

Audience

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

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

Related Documents

For more information, see the following documents in the Oracle Retail Price Management Release 11.0.11 documentation set:

- Oracle Retail Price Management Release Notes
- Oracle Retail Price Management Installation Guide
- Oracle Retail Price Management Data Model
- Oracle Retail Price Management Batch Schedule

Customer Support

<https://metalink.oracle.com>

When contacting Customer Support, please provide the following:

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

Review Patch Documentation

For a base release ("0" release, such as 12.0), Oracle Retail strongly recommends that you read all patch documentation before you begin installation procedures. Patch documentation can contain critical information related to the base release, based on new information and code changes that have been made since the base release.

Oracle Retail Documentation on the Oracle Technology Network

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

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

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

Conventions

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

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

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

A hyperlink appears like this.

Introduction

The information in this document reflects modifications and updates to the *Oracle Retail Price Management 11.0.4 Operations Guide*. (The RPM 11.0.4 Operations Guide is the most recent release of the full Operations Guide for the 11.0 release of RPM.) Using this document in conjunction with that guide provides retailers with a complete overview of the application.

The following changes have been made for the Oracle Retail Price Management 11.0.12 release:

- TaskMDB has changed names to RPMTaskMDB
- The JDK version has changed
- The NewItemLocBatch, MerchExtractKickOffBatch, LocationMoveBatch, statusPageCommandLineApplication, and the WorksheetAutoApproveBatch batch designs have been updated
- There are three new batch designs; injectorPriceEventBatch, PriceChangeAreaDifferentialBatch, and purgeBulkConflictCheckArtifacts.

See the individual chapters of this addendum for specific changes.

For more specific information regarding enhancements and modifications made to the previous Oracle Retail Price Management release, see the Oracle Retail Price Management 11.0.12 Release Notes.

Backend System Administration and Configuration

RPMTaskMDB

RPMTaskMDB is a message driven bean used to facilitate RPM's asynchronous processing capability. Message driven beans act as listeners to specified queues for messages. As soon as a message arrives in the queue, the container triggers execution of this bean.

When a background task is created by RPM, a message is published to the queue as a trigger to start processing of tasks.

By default, publishing is always on. However, to turn off the publishing of tasks to the queue, set the system property "task.publish" to "false".

The jndi names for queueName and ConnectionFactory are specified in retek/system.properties.

Tables Used by RPMTaskMDB

- ... RPM_TASK, RPM_CONFLICT_CHECK_REQUEST_TASK and RPM_CONFLICT_CHECK_REQUEST
Records are inserted into the conflict check request tables (RPM_TASK, RPM_CONFLICT_CHECK_REQUEST_TASK, RPM_LOCATION_MOVE_TASK and RPM_CONFLICT_CHECK_REQUEST). These records are used to instantiate the RPM business object which require a state change involving conflict checking. The task engine instantiates the business object and performs the state change, which initiates conflict checking. Once the state change is completed, the business object is persisted and the task completes.
- ... TASK_ALERT_MODE, TASK_STATE
- ... ALERTS, ALERT_RECEIVER, ALERT_STATUS, ALERT_STATUS_DSC
Tables used for sending alerts to users about task status.

Technical Architecture

Asynchronous Processing

Conflict checking is done in RPM to ensure that rules are followed in order to avert potential pricing problems. Examples of conflict checks include ensuring that a given item/location does not have more than one retail value assigned for a given date, and ensuring that parameters of Regular Price Change, Clearance, and/or Promotions are not causing the retail value of the item/location to go below zero.

The conflict check process in RPM can be either a synchronous or an asynchronous process. If processing is synchronous, it can be kicked off by itself or as part of another action, (for example, approving a price change), but it always blocks the user while it runs. When conflict checking is set as asynchronous, it is a mechanism by which applications can execute long running operations in the background while allowing work to be done simultaneously.

Three system options have been introduced in the RPM application to allow the choice of either synchronous or asynchronous processing during conflict checking. They are as follows:

- ... Price Changes/Clearances
- ... Promotions
- ... Worksheet

Note: By selecting any of the above options, you enable *asynchronous* conflict checking for that functional area.

Synchronous vs. Asynchronous Processing

When conflict checking is done *synchronously*, the system performs conflict checks immediately when the Conflict Check option is selected from the Price Change, Promotions, or Clearances workspaces. It also is performed when state changes occur that require conflict check. The system displays a pop-up dialog showing the conflicts whenever they are found.

Conflict checking is always done synchronously in the following situations:

- ... RSL calls
- ... RIB calls
- ... Batches
- ... Auto-generated price changes

When conflict checking is done *asynchronously*, the conflict check processing happens in a background task or when the system is not busy performing other tasks. You receive an alert once conflict checking is done.

The Conflict Check Results workspace allows you to review the results of background conflict checking for worksheets, price changes, promotions, and clearances. An alert appears in the Conflict column of the worksheet, price change, promotion, or clearance maintenance pane when conflict checking is complete. Refer to the RPM User Guide for instructions to view the results of background conflict checking.

Conflict checking for location moves is expected to operate on extremely large volume of data. It is not acceptable for the user to wait long periods of time for the processing of their request. Thus, conflict checking for location moves is always done asynchronously. Constraints are performed asynchronously at the same time as the conflict check. When you execute an action that kicks off a conflict check, you indicate whether you want constraints checked. If constraints are checked, they are handled just like conflict check errors.

Asynchronous Processing Flow

1. The client requests the application server to perform a process on a business object or set of business objects.
2. The server's application service layer extracts information about the request (type of business object, identifying ID, and requested action).
3. When RPM determines that a task is eligible for asynchronous processing:
 - A JMS message is published to the queue that contains specific information about the task.
 - A record is inserted into the RPM_TASK table. The task-specific information about the task is persisted as a Blob in the table. This task-specific information is a Java object that, when read from the database, is capable of calling an RPM application service to perform asynchronous processing and generate an alert when the processing is completed.
4. As soon as a message arrives in the queue, the container triggers execution of the **RPMTaskMDB**. **RPMTaskMDB** is a message driven bean used to facilitate RPM's asynchronous processing capability. Message Driven Beans acts as listeners to specified queues for messages.
5. The task passes information to the application service layer.
6. The application service layer performs a state transition on the requested business object (for example, approving a particular price change) based on the information passed by the task.
7. The results of this transition (for example, success or failure messages, conflict details) are recorded in the RPM_CONFLICT_CHECK_RESULT table.
8. The application service inserts a record into the ALERT table.
9. The client periodically polls the Alerts application service which looks for new alerts in the ALERTS table.

RPM-Related Java Terms and Standards

Java Development Kit (JDK), version 1.4.2

Standard Java development tools from Sun Microsystems.

Batch Processes

Retailers should refer to these sections in lieu of the sections in the RPM 11.0.4 Operations Guide, “Chapter 7 – Java and RETL Batch Processes”.

Java Batch Processes

This section provides the following:

- An overview of RPM’s batch processing
- A description of how to run batch processes, along with key parameters
- A functional summary of each batch process, along with its dependencies
- A description of some of the features of the batch processes (batch return values, and so on)

Java Batch Process Architectural Overview

The goal of much of RPM’s Java batch processing is to select business objects from the persisted mechanism (for example, a database) by a certain criteria and then to transform them by their state. These RPM Java-based batch processes remove some of the processing load from the real-time online system and are run periodically.

Note the following characteristics of RPM’s batch processes:

- RPM’s batch processes are run in Java. For the most part, batch processes engage in their own primary processing.
- They are not accessible through a graphical user interface (GUI).
- They are scheduled by the retailer.
- They are designed to process large volumes of data, depending upon the circumstances and process.
- They are not file-based batch processes.

Running a Java-Based Batch Process

Java processes are scheduled through executable shell scripts (.sh files). Retek provides each of these shell scripts. During the installation process, the batch shell scripts and the .jar files on which they depend are copied to a client-specified directory structure. See the Installation Guide for details. The batch shell scripts must be run from within that directory structure.

Each script performs the following internally:

- sets up the Java runtime environment before the Java process is run.
- triggers the Java batch process.

To use the scripts, confirm that the scripts are executable (using `ls -l`) and run `chmod +x *.sh` if necessary. The shell scripts take two arguments: username and password. The output can be redirected to a log file (as shown in the example below).

Note: The script `launchRpmBatch.sh` must be modified to include the correct environment information before any of the batch scripts run correctly.

The following is an example of how to use a batch shell script:

```
./locationMoveBatch.sh MyUsername MyPassword > log 2>&1
```

Additional Notes

- All the output (including errors) is sent to the log file.
- The scripts are meant to run in Bash. They have problems with other shells.
- If the scripts are edited on a Windows computer and then transferred to Unix, they may have carriage returns (^M) added to the line ends. These carriage returns (^M) cause problems and should be removed.

Script Catalog

Script	Batch Program Executed
injectorPriceEventBatch.sh	injectorPriceEventBatch
itemLocDeleteBatch.sh	ItemLocDeleteBatch
itemReclassBatch.sh	itemReclassBatch
locationMoveBatch.sh	LocationMoveBatch
merchExtractKickOffBatch.sh	MerchExtractKickOffBatch
newItemLocBatch.sh	NewItemLocBatch
priceChangeAreaDifferentialBatch.sh	PriceChangeAreaDifferentialBatch
priceChangeAutoApproveResultsPurgeBatch.sh	PriceChangeAutoApproveResultsPurgeBatch
priceChangePurgeBatch.sh	PriceChangePurgeBatch
priceChangePurgeWorkspaceBatch.sh	PriceChangePurgeWorkspaceBatch
priceEventExecutionBatch.sh	PriceEventExecutionBatch
priceStrategyCalendarBatch.sh	PriceStrategyCalendarBatch
promotionPurgeBatchbatch.sh	PromotionPurgeBatchbatch
purgeBulkConflictCheckArtifacts.sh	purgeBulkConflictCheckArtifacts
purgeExpiredExecutedOrApprovedClearancesBatch.sh	PurgeExpiredExecutedOrApprovedClearancesBatch
purgeLocationMovesBatch.sh	PurgeLocationMovesBatch
purgeUnusedAndAbandonedClearancesBatch.sh	PurgeUnusedAndAbandonedClearancesBatch
statusPageCommandLineApplication.sh	statusPageCommandLineApplication
worksheetAutoApproveBatch.sh	WorksheetAutoApproveBatch
launchRpmBatch.sh	The retailer does not schedule this script. Other batch programs call this script behind the scenes. Note that this script sets up environment information and takes as a parameter the name of the batch program to run.
zoneFutureRetailPurgeBatch.sh	ZoneFutureRetailPurgeBatch

Scheduler and the Command Line

If the retailer uses a scheduler, arguments are placed into the scheduler.

If the retailer does *not* use a scheduler, arguments must be passed in at the Unix command line.

The Java batch processes are to be called via the shell scripts. These scripts take any and all arguments that their corresponding batch process would take when executing.

Functional Descriptions and Dependencies

The following table summarizes RPM's batch processes and includes a description of each batch process's business functionality.

Batch Processes	Details
InjectorPriceEventBatch	This batch program performs the necessary work to import pricing events (regular price changes, clearance price changes and simple promotions) and optionally submit the events for approval.
ItemLocDeleteBatch	This batch program handles RMS deletions of item locations.
itemReclassBatch	When items are moved from one department/class/subclass to another in the merchandising system, this batch process accordingly sets the correct department/class/subclass for these items in the RPM_FUTURE_RETAIL table.
LocationMoveBatch	This batch process moves locations between zones in a zone group.
MerchExtractKickOffBatch	This batch process builds worksheets in RPM. MerchExtractKickOffBatch.java either creates or updates worksheets based on price strategies and the calendars attached to them.
NewItemLocBatch	This batch program ranges item locations by putting them into the future retail table.
PriceChangeAreaDifferentialBatch	This batch program allows the user to generate and approve Price Changes for selected secondary areas of an Area Differential.
PriceChangeAutoApproveResultsPurgeBatch	This batch process deletes old error message from the price change auto approve batch program.
PriceChangePurgeBatch	This batch process deletes past price changes.
PriceChangePurgeWorkspaceBatch	This batch process deletes abandoned price change workspace records.
PriceEventExecutionBatch	This batch process performs the necessary work to start (regular price change, clearance price change, promotions) and end (price change, promotions) pricing events.
PriceStrategyCalendarBatch	This batch process maintains calendars assigned to price strategies.
PromotionPurgeBatchbatch	This batch process deletes old and rejected promotions.

Batch Processes	Details
purgeBulkConflictCheckArtifacts	This batch program allows you to clean up the working tables in the case that there are environment issues that cause records to be left in these tables.
PurgeExpiredExecutedOrApprovedClearancesBatch	This batch process deletes expired clearances in 'Executed' or 'Approved' statuses.
PurgeLocationMovesBatch	This batch process cleans up expired/executed location moves
PurgeUnusedAndAbandonedClearancesBatch	This batch process deletes unused and rejected clearances.
statusPageCommandLineApplication	
WorksheetAutoApproveBatch	This batch process approves maintain margin strategy worksheets that have not been acted upon by the end of the review period. The strategies must be marked as auto-approve in order to be processed.
ZoneFutureRetailPurgeBatch	This batch deletes past zone/item price change actions.

Batch Process Scheduling

Before setting up an RPM process schedule, familiarize yourself with Batch Schedule document published in conjunction with this release.

Threading and the RPM_BATCH_CONTROL Table

Some RPM batch processes use the RPM_BATCH_CONTROL table, which is a database administrator (DBA) maintained table and is populated by the retailer. This table defines the following:

- The batch process that is to be threaded.
- The number of threads that should be run at a time.
- How much data each thread should process (for example, 2 strategies per thread, 500 item/location/price changes by thread, and so on).

Each batch design later in this chapter states the following in its 'Threading' section:

- Whether the batch process utilizes the RPM_BATCH_CONTROL table.
- Whether or not the batch process is threaded.
- How the batch process is threaded (by strategy, by department, and so on).

Return Value Batch Standards

All batch processes in RPM conform to the Oracle Retail batch standards. They are executed and terminated in the same manner as other batch processes in the Oracle Retail suite of products. The following guidelines describe the return values that RPM's batch processes utilize:

Return Values

- 0 – The function completed without error.
- 1 – A fatal error occurred. The error messages are logged, and the process is halted.

Batch Logging

Relevant progress messages are logged with regard to batch program runtime information. The setting for these log messages is at the Info level in log4j.

For more information, see “Chapter 2 – Backend system administration and configuration” in the RPM 11.0.4 Operations Guide.

Conflict Checking

This section describes conflict checking rules, with an example of a user-defined rule.

Conflict checking is made up of 23 rules that determine whether or not a price event can be approved.

The rules are broken up into three different categories.

- Merge validator conflict checking rules
These rules are “pre-merge,” meaning that the effect of the new price event cannot be added to rpm_future_retail.
- Post-merge conflict checking rules
These rules are “post-merge,” meaning that the effect of the new price event has been added to the RPM_FUTURE_RETAIL table before these 12 rules are run.
- Conflict checking rules controlled by system options
There are three rules that can be turned on or off by disabling or enabling system options in RPM.

Merge Validator Conflict Checking Rules

Merge Validator is the first step in conflict checking. The price events (regular price changes, clearances, or promotions) that are proposed by the user are rejected before they populate the future timeline. This conflict checking is required, and the user cannot choose whether to run these checks.

1. One fixed-price promotion maximum
Only one fixed-price promotion can affect an item/location combination on the timeline. This conflict check verifies that only one fixed-price promotion exists on the same effective day per item/location.
There can be only one fixed-price promotion in the current promotion, or across all existing promotions.
2. Duplicate price change
This rule ensures that the new event does not cause multiple price changes for a given date at any point in time on an item/location timeline. If the date of the price change is equal to the VDATE, it is allowed. This allows multiple emergency price changes for the current day.
3. Duplicate clearance price change
This rule ensures that the new clearance does not cause multiple clearances for a given date at any point on an item/location timeline. If the date of the clearance is equal to the VDATE, it is allowed. This allows multiple emergency clearances for the current day.
4. Multiple clearance events
Reset dates for clearances must be in the past before another clearance event can populate the timeline.

This rule ensures that only one markdown series can exist at any point for an item/location on the future timeline. Only clearance resets that have a date greater or equal to the VDATE are considered.

5. Two promotions maximum for item/location combination

This rule ensures that the new event will not cause more than two promotions to affect an item/location's timeline.

6. One promotion component detail maximum for item/location detail

This rule ensures that the new event does not cause more than one promotion component detail from a single promotion component to affect an item/location at any point on an item/location timeline.

7. Two promotion components maximum for item/location combination

This rule ensures that the event does not cause more than two promotion components from a single promotion to affect an item/location at any point on an item/location timeline.

8. Two exclusions maximum for item/location detail

There can be no more than two approved exclusions on the timeline for an item/location combination.

Post-Merge Conflict Checking Rules (rpm_conflict_query_control Table)

This is the final series in the conflict check process. After the effect of the price event has been added to the RPM_FUTURE_RETAIL table, the validation is done by processing the following rules to ensure that the price event is valid.

In RPM 11.0.8, configuration was added to conflict checking by adding the RPM_CONFLICT_QUERY_CONTROL table. The configuration capabilities are as follows:

- The user can add custom conflict checking rules.
- The user can override the promotion constraint conflict checking rule. The rule that can be turned off is RPM_CC_PROMO_CONSTRAINT.VALIDATE.

Note: It is possible for the user to override any of the remaining 11 conflict checking rules for performance reasons, but this is not supported in base RPM. Turning off any of these rules could cause corrupt data in the RPM_FUTURE_RETAIL table.

9. RPM_CC_NEG_RETAIL.VALIDATE

Future retail cannot be negative. The retail cannot become negative as a result of adding or deleting a price change. Conflict checking is done for any row that is added to RPM_FUTURE_RETAIL.

10. RPM_CC_CLEAR_LT_REG.VALIDATE

Clearance retail must be less than or equal to the regular retail (item/location). The first markdown can be equal to the regular retail, but any additional markdowns must reduce retail.

11. RPM_CC_CLEAR_MKDN.VALIDATE

The clearance markdown must be less than the previous markdown. The new event cannot cause a clearance retail to increase from markdown to markdown at any point on an item/location timeline. The first markdown can make no change, but each additional markdown must be lower than the previous markdown.

12. RPM_CC_CLEARUOM_SELLUOM.VALIDATE

The clearance fixed price change unit of measure (UOM) must be the same as the regular price UOM. For example, if the regular price is \$2 each, a conflict occurs if you try to run a fixed price clearance for \$20 per dozen.

13. RPM_CC_FIXED_CLR_PROM_OVER.VALIDATE

The fixed price clearance cannot overlap with a fixed price promotion if the promotion is defined as “apply to clearance.”

This rule ensures that the new event does not cause a fixed amount clearance to overlap with a promotion that has an “apply to” code of “clearance only” or “regular and clearance” at any point on an item/location timeline.

14. RPM_CC_PROM_LT_CLEAR_REG.VALIDATE

The new selling retail cannot be lower than the promotional retail (simple or complex).

This rule ensures that the new event does not cause the selling or clearance retail to be less than the promotional retail at any point on the item/loc timeline. This rule applies only if the clearance overlap indicator is ‘Y’.

15. RPM_CC_AMOUNT_OFF_UOM.VALIDATE

Amount-off changes cannot change the unit of measure.

This rule ensures that the new price change does not cause a fixed amount change value UOM to differ from the retail UOM at any point on the item/location timeline. For example, it would stop the scenario if the selling retail is \$8 each and the price change is \$2 off per ounce. This conflict check applies to price changes, clearances, and simple/complex promotions.

16. RPM_CC_MULTI_UNIT_UOM.VALIDATE

Multi-unit retail cannot be less than the selling retail.

This rule ensures that the new event does not cause the multi-unit retail to be less than the selling retail, or the selling retail to be more than purchase of two of the multi-units at any point on the item/location timeline. For example, if the single unit retail is \$1 each and the multi-unit retail is \$1.50 for two, the check ensures that the multi-unit retail is greater than the selling unit (yes), and the multi-unit retail is less than the multi-unit quantity times the selling retail: \$1.50 is less than \$1.00 times 2 (yes).

17. RPM_CC_PC_PROM_OV.VALIDATE

A regular price change cannot occur during a promotion.

New price changes cannot overlap with a promotion at any point on the item/location timeline if the price change overlap indicator is ‘N’.

Note: This rule can also be configured by changing the system option.

18. RPM_CC_CL_PROM_OV.VALIDATE

Clearances and promotions cannot overlap.

Clearance price changes and promotional price changes cannot run concurrently unless the clearance overlap indicator is ‘Y’.

Note: This rule can also be configured by changing the system option

19. RPM_CC_PROM_COMP_CNT.VALIDATE

An item/location can exist on more than one promotion at a time.

If Multiple Promotions Ind = 'N', the new promotion must be the only promotion component active within the current promotion, or across other promotions.

20. RPM_CC_PROMO_CONSTRAINT.VALIDATE

Validate new item/location price events against Promotion Constraint.

This is the only rule in the table that can be overridden (turned off). If the rule action is set to 'N', all promotion constraints that are set up in RPM are ignored.

Rules Controlled by System Options

The following rules can be turned on or off by changing system options settings:

21. An item/location can exist on more than one promotion at a given time.

If the indicator is set to 'Yes' (checked), an item can have its retail price affected by more than one promotional discount at a single time in a given location. If the indicator is set to 'No' (unchecked), only one promotional discount can exist at the same time for a given item/location.

22. A regular price change cannot occur during a promotion.

If Price change/Promotion overlap Ind = 'N', a price change cannot overlap with a promotion at any point on the item/loc timeline. If Price change/Promotion overlap Ind = 'Y', then a price change can be approved during a promotion.

23. Clearances and promotions cannot overlap.

If Clearance/Promotion overlap Ind = 'N', a clearance cannot overlap with a promotion at any point on the item/loc timeline. If Clearance/Promotion overlap Ind = 'Y', a clearance can be approved during a promotion.

Adding User-Defined Conflict Checking Rules

The RPM_CONFLICT_QUERY_CONTROL table allows the addition of user-defined rules. When a row is added to the RPM_CONFLICT_QUERY_CONTROL table and a conflict function is implemented that fits the expected prototype, the rule will be executed during all conflict check runs.

The conflict checking functions are executed after the new price event has been added to the RPM_FUTURE_RETAIL table so that the effect of the change can be seen by the rule checking function.

The following is an example of how to add a rule.

Rule definition: No item should sell for less than \$5.

1. Add a row to the rpm_conflict_query_control

```
insert into RPM_CONFLICT_QUERY_CONTROL (
    CONFLICT_QUERY_CONTROL_ID,
    CONFLICT_QUERY_FUNCTION_NAME,
    CONFLICT_QUERY_DESC,
    ACTIVE,
    OVERRIDE_ALLOWED,
    EXECUTION_ORDER,
    BASE_GENERATED,
    BASE_REQUIRED,
    LOCK_VERSION)
select RPM_CONFLICT_QUERY_CONTROL_SEQ.nextval,
    'CUSTOM_RULE.BELOW_FIVE_CHECK',
    '$5 check',
    'Y',
    'Y',
    20,
```

```

        'N',
        'N',
        null
from dual;

```

2. Implement CUSTOM_RULE.BELOW_FIVE_CHECK to fit the expected prototype. The function should take no input parameters. The function should return a CONFLICT_CHECK_ERROR_TBL as an output parameter to hold any error or conflict information. The function should return 0 for failure and 1 for success.

```

CREATE OR REPLACE PACKAGE BODY CUSTOM_RULE AS
-----
FUNCTION BELOW_FIVE_CHECK(IO_error_table      IN OUT CONFLICT_CHECK_ERROR_TBL)
RETURN NUMBER IS

    L_error_key      VARCHAR2(255) := NULL;
    L_error_type     VARCHAR2(255) := NULL;
    L_function       VARCHAR2(61)  := 'RPM_CC_NEG_RETAIL.VALIDATE';

    L_error_rec      CONFLICT_CHECK_ERROR_REC := NULL;
    L_error_tbl      CONFLICT_CHECK_ERROR_TBL := CONFLICT_CHECK_ERROR_TBL();

    cursor c_check is
        select price_event_id,
               future_retail_id
        from rpm_future_retail_gtt gtt
        where gtt.price_event_id NOT IN (select ccet.price_event_id
                                         from table(cast(L_error_tbl as
conflict_check_error_tbl)) ccet)
        and (selling_retail      < 5
            or clear_retail      < 5
            or simple_promo_retail < 5
            or complex_promo_retail < 5);

BEGIN

    if IO_error_table is NOT NULL and
       IO_error_table.count > 0 then
        L_error_tbl := IO_error_table;
    else
        L_error_rec := CONFLICT_CHECK_ERROR_REC(-99999, null, null, null);
        L_error_tbl := CONFLICT_CHECK_ERROR_TBL(L_error_rec);
    end if;

    for rec in c_check LOOP
        L_error_rec := CONFLICT_CHECK_ERROR_REC(rec.price_event_id,
rec.future_retail_id,

RPM_CONFLICT_LIBRARY.CONFLICT_ERROR, 'below_five_check_string');
        if IO_error_table is null then
            IO_error_table := CONFLICT_CHECK_ERROR_TBL(L_error_rec);
        else
            IO_error_table.EXTEND;
            IO_error_table(IO_error_table.COUNT) := L_error_rec;
        end if;
    end LOOP;

    return 1;

EXCEPTION
    when OTHERS then
        L_error_rec := CONFLICT_CHECK_ERROR_REC(null, null,

```

```
RPM_CONFLICT_LIBRARY.PLSQL_ERROR,  
  
SQL_LIB.CREATE_MSG( 'PACKAGE_ERROR' ,  
  
SQLERRM,  
L_function,  
  
to_char(SQLCODE)) );  
    IO_error_table := CONFLICT_CHECK_ERROR_TBL(L_error_rec);  
    return 0;  
END BELOW_FIVE_CHECK;  
-----  
END;  
/
```

3. Define the error string below_five_check_string for the new rule in the Java property file (messages.properties).

Bulk Conflict Checking

This section describes:

- Bulk conflict checking and its impact on performance
- Functional areas affected by bulk conflict checking
- Batch design updates, including any additional parameters, for batch programs that have changed

Overview of Bulk Conflict Checking and Its Impact on Performance

In the previous RPM release, the conflict check engine in RPM is limited to conflict checking only one price event at a time. So when the user sends multiple price events for conflict checking (for example by multi selecting several price events within the Price Change, Clearance or Promotion Component Detail maintenance screen; Price Change Generation for Area Differential; Worksheet approval), RPM would loop through this collection of price events and does the conflict checking for each price event one at a time.

The fact that running the conflict checking for 100 price events with 10 item/locations in each price event took a lot of more time compared to running the conflict checking for one price event with 1000 item/locations proved that there was a great deal of overhead calling the conflict checking process multiple times.

Allowing RPM to do Conflict Checking for a collection of Price Events at a time improves RPM performance, especially in this functionality:

- Price Change generation for Area Differential.
- Worksheet approval.
- Location Move execution (with the new requirement to adjust the Item/Location selling retail following the new zone selling retail).
- External system interface.
- General Price Event approval (i.e., when the user is using the UI to multi select a large number of price events and trying to approve them).

In this release, RPM has been modified so that its Conflict Checking engine is able to process in bulk (that is, multiple price events at a time). The approach taken to modify RPM to be able to do conflict checking for a collection of price events was based on the assumptions below:

- The collection of price events is of the same type (for example, Price Change, Clearance, Simple Promotion, Threshold Promotion and Buy Get Promotion).

- The item/locations that are affected by each price event in the collection are not overlapping. If they are overlapping, then the price events are split into several smaller collections with price events which item/location are not overlapping. These new collections are then conflict checked separately one collection at a time.
- If RIB is turned on, and there is some problem during the publishing, then the whole collection will be rolled back.
- Failures in the below areas (which are executed in the same transaction of the conflict checking process) result in the entire collection in a thread (see explanation on Threading mechanism below) being rolled back
 - Purge of old Future Retail records
 - Population of the payload tables
 - Zone Future Retail processing (for primary zone price changes).
 - Unexpected error during the transaction (e.g. Table Locking or other Database error).

To illustrate how the conflict checking engine processes the collection of price events, please refer to the example below.

Assume that one user sent this collection of price changes to RPM to be approved:

Price Change	Effective Date	Item / Location	Status
PC 1	9/7/07	Item 1 / Zone 1	WORKSHEET
PC 2	9/28/07	Item 1 / Location 1	WORKSHEET
PC 3	9/15/07	Item 2 / Zone 2	WORKSHEET
PC 4	9/30/07	Item 3 / Zone 1	WORKSHEET

In the above table, Zone 1 contains the following locations:

Zone 1 Locations

Location 1
Location 2
Location 3
Location 4

And Zone 2 contains the following locations:

Zone 2 Locations

Location 5
Location 6
Location 7
Location 8
Location 9

The first process is to separate these price changes into several collections (referred as “sequence”) based on the item/location combination so that there are no item/location overlapping in the sequence. From the example above, it can be seen that the PC 1 and PC2 are overlapping. This collection of price changes is then divided into two sequences:

Sequence	Price Change(s)	Effective Date	Item / Location(s)
1	PC 1	9/7/07	Item 1 / Zone 1
	PC 3	9/15/07	Item 2 / Zone 2
	PC 4	9/30/07	Item 3 / Zone 1
2	PC 2	9/28/07	Item 1 / Location 1

The above sequences are then processed one at a time, to ensure that there will be a database locking issue. Note that for the overlapping price changes (PC1 and PC2), the decision of which price changes should go to the first or second sequence is based on the effective date of the price change.

To further make this conflict checking process runs faster, the sequences are divided into several threads. These threads within the sequence are processed simultaneously. To decide the number of threads within the sequence, there is a row in the RPM_BATCH_CONTROL table with PROGRAM_NAME column equals to **'com.retek.rpm.app.bulkcc.service.BulkConflictCheckAppService'**.

The value of the THREAD_LUW_COUNT column decides the maximum number of item/location in the thread. The clients can adjust this number (based on the number of processors, size of the processors, and size of the datasets being processed, among other things) to optimize this conflict checking process in their environment. The default value that is set right now is 10,000.

From the example of the price changes above, if the THREAD_LUW_COUNT is set to 10 then the sequences will be divided into threads similar to the table below:

Processed First: Sequence One:

Sequence	Thread	Price Change	Item / Location
1	1	PC 1	Item 1 / Location 1
			Item 1 / Location 2
			Item 1 / Location 3
			Item 1 / Location 4
1	1	PC 3	Item 2 / Location 5
			Item 2 / Location 6
			Item 2 / Location 7
			Item 2 / Location 8
			Item 2 / Location 9
1	2	PC 4	Item 3 / Location 1
			Item 3 / Location 2
			Item 3 / Location 3
			Item 3 / Location 4

Processed Second: Sequence Two:

Sequence	Thread	Price Change	Item / Location
2	1	PC 2	Item 1 / Location 1

Even though the Item3/Location 1 of PC 4 can fit into Thread 1 of sequence 1, there is an additional rule that prevents that. This rule is **“Item/locations of one price change cannot be processed across multiple threads”**.

InjectorPriceEventBatch Batch Design

Overview

The price events injecting batch process (InjectorPriceEventBatch.java) performs the necessary work to import pricing events (regular price changes, clearance price changes and simple promotions) and optionally submit the events for approval.

Usage

Use the following command to run the job:

```
InjectorPriceEventBatch user_name password status=status_value
event_type=event_type_value polling_interval=polling_interval_value
```

Where:

- user_name – a required argument. The user id of a valid RPM user.
- password – a required argument. The password that confirms credentials for the user.
- status_value – an optional argument. Defines the status for the imported data to process. The valid options are N (New), E (Error), W (Worksheet) or F (Failure) - N (New) is the default.
- event_type_value – an optional argument. Defines the type of pricing event to process. The valid options are PC (price change), CL (clearance) or SP (simple promo) - PC (price change) is the default.
- polling_interval_value – an optional argument. Defines the interval in seconds for the batch to verify if conflict checking is complete. Valid diapason for the interval is 1 to 1000 – 10 seconds is a default.

Please note that the first mandatory arguments (username and password) have a predefined position and order in the list of arguments. The mandatory arguments should be the first in the list of arguments before the optional arguments. The optional arguments may be present in any order after the mandatory arguments. Each optional argument should be preceded by the argument identifier. So to define that the batch needs to process clearances the event type would be defined as

```
event_type=PC
```

Examples

- Only mandatory arguments are defined by the user

```
InjectorPriceEventBatch alain.freon retek
```

The batch processes price changes from the staging table in New status, checking the approval process for completion every ten seconds.

- All arguments are defined by the user

```
InjectorPriceEventBatch alain.freon retek event_type=CL status=W
polling_interval=300
```

The batch processes clearances from the staging table in Worksheet status, checking the approval process for completion every five minutes (300 seconds).

Additional Notes

The batch should be run by means of shell script *injectorPriceEventBatch.sh*.

Details

The batch program imports regular price changes, clearance, and simple promotions that have been generated by an external to RPM application. The batch doesn't make any assumptions about the source of the price event data. The only requirement for the data is to adhere to the predefined importing data format. The contract on the incoming data is defined by the structure of the staging tables the batch depends on. The staging tables work as the interface point between RPM and the external system providing the data. All the necessary data transformation possibly required to accommodate RPM price event data requirements should be done before populating the staging tables and is a client responsibility.

If there is a mistake in the data file and bulk numbers of price events are created with the data, (Example: wrong date or the wrong retail) we need to have the ability to rollback all data in order to re-process the file with the right values.

Steps to change a Price Change/Clearance and Promotion from Approved to Worksheet Status:

1. Set up the data in the appropriate staging table with item/locs, status of N and auto_approve = 1.
2. Run the price injector batch, status parameter of N
3. Price events are created in UI and staging table is updated with status of A(Approved).
4. To set the approved events back to worksheet, leave the same item/locs in the table from step 1. Run the price injector batch, status parameter of A. This means we want to execute all of the price events with status of A, and doing so will set the price event back to worksheet.
5. Verify the price events are set back to worksheet in the staging and the UI.

Importing Staged Price Changes

Staged price changes data should be placed by the system administrator into RPM_STAGE_PRICE_CHANGE table. The table has the structure similar to that of RPM_PRICE_CHANGE but with some limitations. The limitations are:

- no price change exceptions are allowed
- no parent exceptions are allowed
- no vendor funding allowed
- no deals allowed

Besides field carrying data payload the table holds fields facilitating data processing.

- *Auto approve indicator* defines whether the processing batch should attempt to approve the price change after successfully importing the data
- *Status* defines the current state of the data in the staging table. The status should not be confused with the status of the price event in RPM, even though there are some correlations. Possible statuses are **N** (New), **W** (Worksheet), **A** (Approved), **E** (Error) and **F** (Failed). Initial data should be created in the New status. The rest of the statuses are the result of data lifecycle. The records in all but Approved statuses are eligible for processing by the batch. New status indicates that the data hasn't been

processed yet. Worksheet status indicates that the data has been processed and has successfully been imported into RPM. It also indicates that at the time of import the approval wasn't required. Error status indicates that the data has been processed but import has failed due to invalid data. Data administrator can correct the data and the processing can be retried. Failed status indicates that the data has been successfully imported but some conflicts have been encountered. Data administrator can correct the data (for example to change the effective date of the event to eliminate the conflict) and the processing can be retried. Approved status indicates that the data has been successfully imported and successfully approved without any conflicts.

- *Error message* holds the message for the first error encountered while importing data. The field actually holds an error message key rather than the actual error message. When conflict was encountered on approval attempt the field holds "CONFLICT_EXISTS" key.
- *Process id* uniquely identifies the batch run. The field is populated for records in all statuses but New.
- *Price change display id* will be populated only on successful import of the data.

Importing Staged Clearances

Staged clearance data should be placed by the system data administrator into RPM_STAGE_CLEARANCE table. The table has the structure similar to that of RPM_CLEARANCE but with some limitations. The limitations are:

- no clearance exceptions are allowed
- no reset records are allowed
- no vendor funding allowed
- no deals allowed

Besides field carrying data payload the table holds fields facilitating processing.

- *Auto approve indicator* defines if the processing batch should attempt to approve the clearance after successfully importing the data
- *Status* defines the current state of the data. Possible statuses are **N** (New), **W** (Worksheet), **A** (Approved), **E** (Error) and **F** (Failed). Initial data should be created in the **N** (New) status. The rest of the statuses are the result of data lifecycle. The records in all but Approved statuses are eligible for processing. New status indicates that the data hasn't been processed yet. Worksheet status indicates that the data has been processed and has successfully been imported into RPM. It also indicates that at the time of import the approval wasn't required. Error status indicates that the data has been processed but import has failed due to invalid data. Data administrator can correct the data and the processing can be retried. Failed status indicates that the data has been successfully imported but some conflicts have been encountered. Data administrator can correct the data and the processing can be retried. Approved status indicates that the data has been successfully imported and successfully approved without any conflicts.
- *Error message* holds the message for the first error encountered while importing data. The field actually holds an error message key rather than the actual error message. When conflict was encountered on approval attempt the field holds "CONFLICT_EXISTS" key.
- *Process id* uniquely identifies the batch run. The field is populated for records in all statuses but New.
- *Clearance display id* will be populated only on successful import of the data.

Importing Staged Simple Promotions

Staged simple promo data should be placed by the system data administrator into `RPM_STAGE_PROMO_COMP_SIMPLE` table. The table has the structure similar to that of `RPM_PROMO_COMP_SIMPLE` but with the limitation that no exceptions are allowed.

Besides field carrying data payload the table holds fields facilitating processing.

- *Auto approve indicator* defines if the processing batch should attempt to approve the promotion after successfully importing the data
- *Status* defines the current state of the data. Possible statuses are **N** (New), **W** (Worksheet), **A** (Approved), **E** (Error) and **F** (Failed). Initial data should be created in the **N** (New) status. The rest of the statuses are the result of data lifecycle. The records in all but Approved statuses are eligible for processing. New status indicates that the data hasn't been processed yet. Worksheet status indicates that the data has been processed and has successfully been imported into RPM. It also indicates that at the time of import the approval wasn't required. Error status indicates that the data has been processed but import has failed due to invalid data. Data administrator can correct the data and the processing can be retried. Failed status indicates that the data has been successfully imported but some conflicts have been encountered. Data administrator can correct the data and the processing can be retried. Approved status indicates that the data has been successfully imported and successfully approved without any conflicts.
- *Error message* will hold the message for the first error encountered while importing the data. The field will actually hold an error message key rather than the actual error message.
- *Process id* uniquely identifies the batch run. The field is populated for records in all statuses but New.

Main Steps Taken by the Batch

- Generate the pricing events (import pricing events). This step will load data from the staging table (actual table depends on the event type specified as a batch argument). The batch will validate the data based on RPM validity rules. If at least a single field for a record isn't valid, the record will be rejected. The first encountered error will be reported (`ERROR_MESSAGE` column in the staging table will be populated) and the status will be set to **ERROR**. The records with multiple incorrect data fields would need to be processed multiple times unless corrected by the data administrator all at once. If no pricing events have been generated the batch will terminate at this stage. All records in the staging table that match the run argument criteria will be processed at once. At the same time all records are independent in terms that data errors encountered on one record will not impact processing of other records.
- If at least a single pricing event was generated the batch will proceed with an optional approval step. For the batch to attempt the approval on a record, the data administrator populating the staging data should set auto approval flag on the record to **ON** (`AUTO_APPROVE_IND` should be set to 1). In this case the batch will attempt to approve the price event. This involves conflict checking. This process can take quite a long time depending on the volume of data. This step will end only when the approval process will report completion of all threads responsible for conflict checking. Depending on the volume of data, the interval the batch uses to poll conflict checking logic for completion should be adjusted accordingly.
- If auto approval was not requested or approval process failed, the data will be left in appropriate RPM table in **Worksheet** status. If the approval was successful, then the data will be in **Approved** status. The status on the staging table after approval step will be either **Approved** or **Failed**, depending on how the process terminated.

- The conflict checking logic will not purge intermediate data to allow the batch to inquire on the state of the conflict checking process. After it was determined that the CC logic is complete the batch will purge CC intermediate data.
- The final step for the batch process is to generate report. The report gives the system administrator statistics on the batch run. The following information will be provided:
 - What initial status to batch was processing
 - What was event type
 - How many records have been imported
 - How many records required approval
 - How many records have been successfully approved

Assumptions and Scheduling Notes

- It is assumed that a single instance of the batch will be running at a time so a single event type will be processed at a time.
- Approved pricing events on the staging tables can't be processed again.

Primary Tables Involved

- RPM_STAGE_PRICE_CHANGE and RPM_PRICE_CHANGE
- RPM_STAGE_CLEARANCE and RPM_CLEARANCE
- RPM_STAGE_PROMO_COMP_SIMPLE and RPM_PROMO_COMP_SIMPLE and RPM_PROMO_COMP_DETAIL

These tables provide data to be imported by the batch. Each record is independently processed.

Threading

This program is not threaded by itself. The main batch logic is executed as part of a single thread. At the same time the batch rely for approval on a multi-threaded conflict checking logic. The batch will poll this logic with the interval defined by *polling_interval* parameter to identify the completion point.

LocationMoveBatch Batch Design

Overview

The LocationMoveBatch program moves locations between zones in a zone group.

Usage

The following command runs the LocationMoveBatch job:

```
LocationMoveBatch userid password <RETRY_NUMBER>
```

Where the first argument is the user id and the second argument is the password. The third argument is optional and starts the batch in RESTART mode if the original batch run has failures. The RETRY_NUMBER argument is the maximum number of retry attempts for processing error records.

Detail

The batch looks for scheduled zone location move and updates the zone structure tables with the new zone structure.

- Remove the location from the old zone.
- Add the location to the new zone.

Update FUTURE_RETAIL table to reflect the location move.

- Price events (standard price change, clearance price change, promotion) scheduled for item/locations effected by the move at the old zone level are removed from FUTURE_RETAIL
- Price events (standard price change, clearance price change, promotion) scheduled for item/locations effected by the move at the new zone are added to FUTURE_RETAIL
- Conflict checking is run on FUTURE_RETAIL after event from the old zone are removed and events from the new zone are added. If conflicts are encountered during conflict checking, exceptions / exclusions are pulled off the conflicting event.

Report any exception / exclusions that were created during the FUTURE_RETAIL update process. Changes made are held on:

- RPM_LOC_MOVE_PRC_CHNG_EX
- RPM_LOC_MOVE_CLEARANCE_EX
- RPM_LOC_MOVE_PROMO_COMP_DTL_EX

Update the status of the location move to 'Executed.'

If errors occur during processing, the Location Move Batch will finish and log the errors to the RPM_LOCATION_MOVE_ERROR table. Users can use this table to ascertain the source of the problems, correct them, and then start the Location Move batch Restart mode. The batch running in Restart mode will attempt to apply the price events for the location move to FUTURE_RETAIL.

To start the batch in Restart mode, an additional parameter (third parameter) must be passed into the batch program indicating the maximum number of retries. If this parameter is set, then the batch is driven off of the RPM_LOCATION_MOVE_ERROR table for records that have less retry attempts than the maximum number of retries flag (RETRY_NUMBER). If there is still an error in processing in Restart mode, the error record is updated and the RETRY_NUMBER is incremented.

Assumptions and Scheduling Notes

LocationMoveBatch must run before the following programs:

- PriceEventExecutionBatch,
- MerchExtractKickOffBatch

Primary Tables Involved

RPM_FUTURE_RETAIL

Threading

This program is threaded. Each location move request is given its own thread.

MerchExtractKickOffBatch Batch Design

Overview

The MerchExtractKickOffBatch.java batch program builds worksheets in RPM. MerchExtractKickOffBatch.java either creates or updates worksheets based on price strategies and the calendars attached to them.

Usage

The following command runs the MerchExtractKickOffBatch job:

```
MerchExtractKickOffBatch userid password <mode>
```


Where the first argument is the user id and the second argument is the password. The optional third argument can be used to split the processing into three components: pre-process, process and post-process. The valid values for this argument are: PRE, PROCESS, POST and ALL. ALL is the default value for the mode argument when none is provided.

The program is split up into sections for performance and functional reasons. The population of the RPM_PRE_ME tables in the setup section allows access to the largest RMS tables in the most performant manner. The splitting up of the worksheet creation section ensures that a worksheet will not be reprocessed in the case of a failure in a different worksheet. The splitting up of a post process helps to avoid locking issues.

Detail

Setup: (included in modes: ALL and PRE) clean up expired worksheets and prepare for creation of new worksheets.

- Delete worksheets that are at the end of their review period.
- Get list of all strategies that need to be processed today. Create copies of the strategies as needed.
- Determine what strategies need to be grouped together based on the RPM_DEPT_AGGREGATION. WORKSHEET_LEVEL.
- Stages date in RPM_PRE_ME_AGGREGATION, RPM_PRE_ME_ITEM_LOC, RPM_PRE_ME_COST, and RPM_PRE_ME_RETAIL. This is done for performance reasons. This allows the program to access large tables in an efficient as possible manner.

Worksheet Creation: (included in modes: ALL and PROCESS)

- Start threads based on the values in RPM_BATCH_CONTRL for MerchExtractKickOffBatch.java.
- Call RPM_EXT_SQL, a PL/SQL package, to extract RPM information. The package is called at the strategy and RPM_DEPT_AGGREGATION. WORKSHEET_LEVEL. level. It pulls large amounts of data from various RMS tables and populates the RPM_WORKSHEET_DATA table. The RPM_MERCH_EXTRACT_CONFIG table is used to exclude certain families of data from being included in the population. If this table is not populated all values are included in the population of RPM_WORKSHEET_DATA.
- For each RPM_WORKSHEET_DETAIL record created perform the following:
 - Use the price strategy to propose a retail value.
 - Apply candidate rules.
 - Apply price guides.

Potential reasons item/locations do not make it into a worksheet:

- The item/location falls under an exclusion type candidate rule.
- The item/location does not have a cost on RMS's FUTURE_COST table.
- The item's market basket codes vary across locations in a zone.
- The item's link code varies across locations in a zone.
- If a link code is identified on an item/location, and there is any item within that link code (at that location) that has not been brought into the worksheet, all of the item/locations with that link code are excluded from the worksheet.
- The item's selling unit of measure varies across locations in a zone.
- The item is part of an area differential item exclusion.
- Item/locations in a single link code have varying selling unit of measures.

If an item does not make it into a worksheet, a row is inserted into the RPM_MERCH_EXTRACT_DELETIONS table for each item location along with a reason that the item location was not included in the worksheet.

Post process: (included in modes: ALL and POST)

- Update the COMP_PRICE_HIST table. This logic needs to be in a post process to avoid locking issues as multiple threads can share competitive pricing information.

Assumptions and Scheduling Notes

The following programs must run before PriceStrategyCalendarBatch:

- PriceStrategyCalendarBatch
- LocationMoveBatch

Primary (RPM) Tables Involved

- RPM_WORKSHEET_STATUS
- RPM_WORKSHEET_DATA
- RPM_STRATEGY
 - RPM_STRATEGY_CLEARANCE
 - RPM_STRATEGY_CLEARANCE_MKDN
 - RPM_STRATEGY_COMPETITIVE
 - RPM_STRATEGY_DETAIL
 - RPM_STRATEGY_MARGIN
 - RPM_STRATEGY_REF_COMP
 - RPM_STRATEGY_WH
- RPM_AREA_DIFF
 - RPM_AREA_DIFF_EXCLUDE
 - RPM_AREA_DIFF_PRIM
 - RPM_AREA_DIFF_WH
- RPM_CALENDAR
 - RPM_CALENDAR_PERIOD
- RPM_CANDIDATE_RULE
 - RPM_CONDITION
 - RPM_VARIABLE
 - RPM_VARIABLE_DEPT_LINK
- RPM_PRICE_GUIDE
 - RPM_PRICE_GUIDE_DEPT
 - RPM_PRICE_GUIDE_INTERVAL

Threading

MerchExtractKickOffBatch.java is threaded. The RPM_BATCH_CONTROL table must include a record for MerchExtractKickOffBatch.java for it to run in threaded mode.

MerchExtractKickOffBatch.java is threaded by strategies and the RPM_DEPT_AGGREGATION. WORKSHEET_LEVEL setting.

PL/SQL Interface Point

- Package: RPM_EXT_SQL

NewItemLocBatch Batch Design

Overview

The NewItemLocBatch program replaces the Item/Location Creation RIB message. It ranges item locations by putting them into the future retail table. Item and location are fed to this program via the RPM_STAGE_ITEM_LOC table, which is populated by an RMS process.

Usage

The following command runs the NewItemLocBatch job:

```
NewItemLocBatch <userid> <password> [<status> <logical commit count>]
```

Where the first argument is the user id and the second argument is the password. The last two arguments are optional and direct the application as to what “status” (the rows in the stage table with a status of N or E (new or error) and logical unit of work per thread to process. If none is indicated the logical unit of work is used for processing new rows.

Since this batch is a replacement for the Item/Location Creation RIB message, the following steps should be followed in preparation for using this batch in place of the RIB:

- Delete existing records from table RPM_STAGE_ITEM_LOC
- Enable the new trigger RMS_TABLE_RPM_ITL_AIR on table ITEM_LOC.
- Stop the listener for Item/Location Creation RIB messages:
 - Login to Websphere admin console
 - Select the RIBforRPM server from *Servers* → *Application Servers*.
 - Click the “Message Listener Service” link.
 - Click the “Listener Ports” link.
 - Click the checkbox next to “ItemLocToRPMPort”
 - Click the “Stop” button. The listener is now stopped.
 - Click the “ItemLocToRPMPort” link to configure the port.
 - Select “Stopped” from the “Initial State” combo-box.
 - Click “OK”
 - Restart the RIBforRPM app server and verify that “ItemLocToRPMPort” is stopped.
- Delete JMS subscriber in Egate
 - Login to the Egate Schema Manager
 - Click the “JMS Administrator” button in the toolbar.
 - Expand item “etItmLocFromRMS”
 - Right click on the RPM subscriber (it should have “RPM” in its name) and select “delete subscriber”
- Add a record to table RPM_BATCH_CONTROL with PROGRAM_NAME of ‘com.retek.rpm.batch.NewItemLocBatch’ to control threading.

To reconfigure the system to process Item/Location Creation messages through the RIB, follow these steps:

- Start the listener for Item/Location Creation RIB messages:
 - Login to Websphere admin console
 - Select the RIBforRPM server from *Servers* → *Application Servers*.
 - Click the “Message Listener Service” link.

- Click the “Listener Ports” link.
- Click the checkbox next to “ItemLocToRPMPort”
- Click the “Start” button. The listener is now started.
- Click the “ItemLocToRPMPort” link to configure the port.
- Select “Started” from the “Initial State” combo-box.
- Click “OK”
- Restart the RIBforRPM app server and verify that “ItemLocToRPMPort” is stopped.
- Create JMS subscriber in Egate
 - No action is needed.
- Disable the new trigger RMS_TABLE_RPM_ITL_AIR on ITEM_LOC.
- Run the new batch (NewItemLocBatch.sh) one time to process any records remaining in RPM_STAGE_ITEM_LOC.

Detail

The batch selects rows from the stage table and updates the FUTURE_RETAIL table to reflect the new item/location combination. If any approved price changes/promotions/clearances exist at a parent/zone level that encompasses the new item/location, these are also added to the FUTURE_RETAIL table for the new item/location.

Assumptions and Scheduling Notes

This batch may be run ad-hoc. However, it should be noted that the item/locations to be processed will only inherit pricing events that are approved (or active) at the time of the run.

Primary Tables Involved

- RPM_STAGE_ITEM_LOC
- RPM_STAGE_ITEM_LOC_CLEAN
- RPM_FUTURE_RETAIL

Threading

This program is threaded. If no row exists in the RPM_BATCH_CONTROL table for com.retek.rpm.batch.NewItemLocBatch, then the application is executed with one thread and transactions are committed for each item-loc combination.

Bulk Conflict Checking

This program now utilizes the new bulk conflict checking framework inside of RPM. Each thread that is spawned by the RPM batch threading framework (threaded by item/loc) may spawn other threads (by pricing events) during its processing. See the Bulk Conflict Checking documentation for more details.

Processing Stage Rows in Error Status:

This program is set up to re-process (re-attempt) rows that end up in error status. In the event that an error occurs during the processing of “new” status rows, the program should update the status on the stage table with “E” along with an error message. Once the error has been fixed, you can re-run this program with status “E” in an attempt to get the item/loc into RPM.

PriceChangeAreaDifferentialBatch Batch Design

Overview

The Price Change Area Differential batch process (PriceChangeAreaDifferentialBatch.java) allows the user to generate and approve Price Changes for selected secondary areas of an Area Differential.

Usage

Use the following command to run the job:

```
PriceChangeAreaDifferentialBatch password user_name
```

Where:

- user_name is a required argument. The user id of a valid RPM user.
- password is a required argument. The password that confirms credentials for the user.

Additional Notes

The batch should be run by means of the shell script named priceChangeAreaDifferentialBatch.sh.

Details

The batch program was changed for RPM 11.0.11.2:

- The Bulk Conflict Checking engine is used to conflict check the generated Price Changes
- Instead of the batch process spawning multiple threads to do the approval, the threading is done by the Bulk Conflict Checking engine

Assumptions and Scheduling Notes

Only one instance of the batch (per database) may be run at a time.

Primary Tables Involved

- RPM_AREA_DIFF
- RPM_PRICE_CHANGE
- RPM_FUTURE_RETAIL (if auto-approve)

purgeBulkConflictCheckArtifacts Batch Design

Overview

The purgeBulkConflictCheckArtifacts program cleans up the working tables used by Bulk Conflict Checking engine.

Usage

The following command runs the purgeBulkConflictCheckArtifacts job:

```
purgeBulkConflictCheckArtifacts userid password
```

Where the first argument is the user id and the second argument is the password.

Detail

The current release of RPM comes with the Bulk Conflict Checking engine. This engine is using several working table in order to do its process. In normal condition, these tables

are supposed to be deleted at the end of the Bulk Conflict Checking process. But if there is any environment issue, it is possible that there are some records left in these tables. This batch program will clean up those working table. This batch will make sure that system has a clean set of working tables for Bulk Conflict checking for the next day. Users using the application with too many records left in these working tables could deter the performance of Bulk Conflict Checking.

Assumptions and Scheduling Notes

`purgeBulkConflictCheckArtifacts` needs to run at the end of all other batch programs.

Primary Tables Involved

- `RPM_BULK_CC_PE`
- `RPM_BULK_CC_PE_SEQUENCE`
- `RPM_BULK_CC_PE_THREAD`
- `RPM_BULK_CC_PE_IL`

statusPageCommandLineApplication Batch Design

The status page batch program (`statusPageCommandLineApplication.sh`) was enhanced to perform some data checks, to verify that some of the assumptions that the application makes about the data are not violated. The checks are done with SQL counts; each check should return 0 rows.

These are the data checks that are performed:

- Missing department aggregations—When departments are created in RMS, a row should be inserted into the `RPM_DEPT_AGGREGATION` table.
- Missing primary zone groups—Each merchandise hierarchy (department or lower) should have a row in the `RPM_MERCH_RETAIL_DEF` table.
- Missing item/locations from future retail—When an item is ranged to a location in RMS, a row should be inserted into the `RPM_FUTURE_RETAIL` table.
- Duplicate future retail—There should only be one row in the the `RPM_FUTURE_RETAIL` table per item, location, and action date.

The new command usage is as follows:

```
statusPageCommandLineApplication.sh username password [phase-choice]
[max-rows-choice]
```

Valid values for *phase-choice* are as follows:

S	System check only
D	Data integrity check only
B (default)	Both

The value specified for *max-rows-choice* is the maximum row count for the query. By default, the query is run for the full count.

For example:

```
./statusPageCommandLineApplication.sh alain.freon retek S
```

Here is sample output of the batch program.

```
Performing System Check
The following RpmRibMessageStatusException is normal.
We need to throw an exception to ensure that the test messages are rolled back.
10:30:04,599 ERROR [ServiceAccessor] InvocationTargetException received on a
service call...
java.lang.reflect.InvocationTargetException
```

```

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:79)
at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java(
Compiled Code))
at java.lang.reflect.Method.invoke(Method.java(Compiled Code))
at org.apache.commons.beanutils.MethodUtils.invokeMethod(MethodUtils.java:216)
at
com.retek.platform.service.ServiceAccessor.callRemoteMethod(ServiceAccessor.java:3
00)
at
com.retek.platform.service.ServiceAccessor.remoteTransaction(ServiceAccessor.java:
485)
at
com.retek.platform.service.ServiceAccessorProxy.invoke(ServiceAccessorProxy.java:5
1)
at $Proxy4.performRibMessageCheck(Unknown Source)
at com.retek.rpm.statuspage.RpmRibMessageCheck.execute(RpmRibMessageCheck.java:25)
at com.retek.rpm.statuspage.StatusPageCheck.runTest(StatusPageCheck.java:15)
at
com.retek.rpm.statuspage.StatusPageProcessor.execute(StatusPageProcessor.java:19)
at
com.retek.rpm.statuspage.StatusPageCommandLineApplication.performAction(StatusPage
CommandLineApplication.java:80)
at
com.retek.rpm.statuspage.StatusPageCommandLineApplication.main(StatusPageCommandLi
neApplication.java:65)
Caused by:
<com.retek.rpm.app.statuspage.service.RpmRibMessageStatusException>
<message>
    No cause associated
</message>
</com.retek.rpm.app.statuspage.service.RpmRibMessageStatusException>

at
com.retek.rpm.app.statuspage.service.StatusPageAppServiceImpl.performRibMessageChe
ck(StatusPageAppServiceImpl.java:71)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java(Compiled
Code))
at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java(
Compiled Code))
at java.lang.reflect.Method.invoke(Method.java(Compiled Code))
at org.apache.commons.beanutils.MethodUtils.invokeMethod(MethodUtils.java(Compiled
Code))
at
com.retek.platform.service.ServiceCommandImpl.execute(ServiceCommandImpl.java(Comp
iled Code))
at
com.retek.platform.service.impl.CommandExecutionServiceEjb.executeCommand(CommandE
xecutionServiceEjb.java(Compiled Code))
at
com.retek.platform.service.impl.EJSRemoteStatelessCommandExecutionService_76208b17
.executeCommand(Unknown Source)
at
com.retek.platform.service.impl._EJSRemoteStatelessCommandExecutionService_76208b1
7_Tie.executeCommand__com_retek_platform_service_ServiceCommand(_EJSRemoteStateles
sCommandExecutionService_76208b17_Tie.java(Compiled Code))

```

```

at
com.retek.platform.service.impl._EJSRemoteStatelessCommandExecutionService_76208b1
7_Tie._invoke(_EJSRemoteStatelessCommandExecutionService_76208b17_Tie.java(Compile
d Code))
at
com.ibm.CORBA.iiop.ServerDelegate.dispatchInvokeHandler(ServerDelegate.java(Compil
ed Code))
at com.ibm.CORBA.iiop.ServerDelegate.dispatch(ServerDelegate.java(Compiled Code))
at com.ibm.rmi.iiop.ORB.process(ORB.java(Compiled Code))
at com.ibm.CORBA.iiop.ORB.process(ORB.java(Compiled Code))
at com.ibm.rmi.iiop.Connection.doWork(Connection.java(Compiled Code))
at com.ibm.rmi.iiop.WorkUnitImpl.doWork(WorkUnitImpl.java(Compiled Code))
at com.ibm.ejs.oa.pool.PooledThread.run(ThreadPool.java(Compiled Code))
at com.ibm.ws.util.ThreadPool$Worker.run(ThreadPool.java(Compiled Code))
*****
Starting Report
com.retek.rpm.statuspage.RsmServerCheck Passed
*****
Starting Report
com.retek.rpm.statuspage.RpmLoginCheck Passed
*****
Starting Report
com.retek.rpm.statuspage.RpmDataAccessCheck Passed
*****
Starting Report
com.retek.rpm.statuspage.RpmRibMessageCheck Passed REGPRCCHG.REGPRCCHGCRE is ON
*****The above exception indicates that we have passed
*****
*****
Starting Report
com.retek.rpm.statuspage.RpmRibMessageCheck Passed REGPRCCHG.REGPRCCHGMOD is ON
*****The above exception indicates that we have passed
*****
*****
Starting Report
com.retek.rpm.statuspage.RpmRibMessageCheck Passed REGPRCCHG.REGPRCCHGDEL is ON
*****The above exception indicates that we have passed
*****
*****
Starting Report
com.retek.rpm.statuspage.RpmRibMessageCheck Passed CLRPRCCHG.CLRPRCCHGCRE is ON
*****The above exception indicates that we have passed
*****
*****
Starting Report
com.retek.rpm.statuspage.RpmRibMessageCheck Passed CLRPRCCHG.CLRPRCCHGMOD is ON
*****The above exception indicates that we have passed
*****
*****
Starting Report
com.retek.rpm.statuspage.RpmRibMessageCheck Passed CLRPRCCHG.CLRPRCCHGDEL is ON
*****The above exception indicates that we have passed
*****
*****
Starting Report
com.retek.rpm.statuspage.RpmRibMessageCheck Passed PRMPRCCHG.PRMPRCCHGCRE is ON
*****The above exception indicates that we have passed
*****
*****
Starting Report
com.retek.rpm.statuspage.RpmRibMessageCheck Passed PRMPRCCHG.PRMPRCCHGMOD is ON
*****The above exception indicates that we have passed
*****
*****

```



```

*****
Starting Report
com.retek.rpm.statuspage.RpmRibMessageCheck Passed PMPRCCHG.PMPRCCHGDEL is ON
*****The above exception indicates that we have passed
*****
*****
Starting Report
RpmJmsServerCheck Passed

Done.

```

WorksheetAutoApproveBatch Batch Design

Overview

The WorksheetAutoApproveBatch program approves maintain margin strategy worksheets that have not been acted upon by the end of the review period. The strategies must be marked as auto-approve in order to be processed.

Usage

The following command runs the WorksheetAutoApproveBatch job:

```
WorksheetAutoApproveBatch userid password
```

Where the first argument is the user id and the second argument is the password.

Detail

The WorksheetAutoApproveBatch first finds strategies to process. In order to qualify the strategies must meet the below criteria:

- The strategies must be a maintain margin strategy with its auto-approve indicator set.
- The strategies must be associated with a calendar review period that is ending.

For each strategy that qualifies, worksheet detail records are processed. In order to be processed the worksheet detail records must meet the following criteria:

- The worksheet detail must be marked as either undecided or take.
- The worksheet detail record must be represent an actual change in the retail
- The worksheet detail must be in one of the below states:
 - New
 - In progress
 - Pending
 - Submitted
 - Submit rejected
 - Updated

Each worksheet detail that meets the above criteria will have run through the approval logic. The approval logic attempts to create and approve a price change. If the price change cannot be approved, the reason is written to the Conflict Check Results Dialogue. Additionally, area differential logic is executed.

If dynamic area differentials are being used in the system, any secondary area worksheet detail records that exist are also processed. If the secondary area is marked as auto approve, the secondary worksheet detail record goes through the same logic as the original worksheet detail (depending on its state). If the secondary area is not marked as auto approve, the secondary worksheet detail record has a retail proposed for it and will move into new status and become available for review by online users.

After all the worksheet details for the working strategy have been run through their approval logic, the worksheet status is updated to reflect the changes made to the details.

Assumptions and Scheduling Notes

WorksheetAutoApproveBatch can be run ad hoc.

Primary Tables Involved

- RPM_STRATEGY_MAINT_MARGIN
- RPM_WORKSHEET_STATUS
- RPM_WORKSHEET_DATA
- RPM_PRICE_CHANGE
- RPM_CLEARANCE
- RPM_FUTURE_RETAIL
- RPM_AREA_DIFF_PRIM
- RPM_AREA_DIFF
- RPM_MAINT_MARGIN_ERR
- RPM_MAINT_MARGIN_ERR_DTL

Threading

This program is threaded but does not use the RPM_BATCH_CONTROL table. Please see the documentation for Bulk Conflict Check for thread configuration details.