Oracle® Retail Price Management

Operations Guide
Release 12.0.8

July 2008

ORACLE

Oracle® Price Management Operations Guide, Release 12.0.8

Copyright © 2008, Oracle. All rights reserved.

Primary Author: Susan McKibbon

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, report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in the 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

Oracle Retail Operations Guides describe behind-the-scenes processing and include the following information.

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

Audience

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

- System analysts and system operation personnel who
  - are looking for information about RPM's processes internally or in relation to the systems across the enterprise.
  - operate RPM on a regular basis.
- Integrators and implementation staff who have the overall responsibility for implementing RPM into their enterprise.
- Business analysts who are looking for information about processes and interfaces to validate the support for business scenarios within RPM and other systems across the enterprise.

Related Documents

For more information, see the following documents in the Oracle Retail Price Management Release 12.0.8 documentation set:

- Oracle Retail Price Management Installation Guide
- Oracle Retail Price Management Release Notes
- Oracle Retail Price Management Data Model
- Oracle Retail Merchandising 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 taken

Review Patch Documentation

For a base release (".0" release, such as 12.0), Oracle Retail strongly recommends reading all patch documentation before beginning 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 explains 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.

Third-Party Open-Source Applications

Oracle Retail Security Manager includes the following third-party open-source applications:

Software Provider: Object Web
Software Name: ASM
Software Version: 1.4.3
Jar File Name: asm-1.4.3.jar
Provider Web Site: http://forge.objectweb.org/projects/asm

Software Provider: Bouncy Castle
Software Name: JCE Provider
Software Version: JDK 1.4 v1.2.4
Jar File Name: bcprov-jdk14-124.jar
Provider Web Site: http://www.bouncycastle.org

Software Provider: Bean Shell
Software Name: Bean Shell
Software Version: 2.0
Jar File Name: bsh-2.0b1.jar
Provider Web Site: http://www.beanshell.org/

Software Provider: Intalio Inc., and others
Software Name: Castor
Software Version: 0.9.5.2
Jar File Name: castor-0.9.5.2.jar
Provider Web Site: http://www.castor.org/

Software Provider: Apache Software Foundation
Software Name: cglib
Software Version: 2.0.2
Jar File Name: cglib-2.0.2.jar
Provider Web Site: http://cglib.sourceforge.net/

Software Provider: Apache Software Foundation
Software Name: org.apache.commons.beanutils-bean-collections
Software Version: 1.6
Jar File Name: commons-beanutils-bean-collections.jar
Provider Web Site: http://jakarta.apache.org/commons/

Software Provider: Apache Software Foundation
Software Name: org.apache.commons.beanutils
Software Version: 1.6
Jar File Name: commons-beanutils-core.jar
Provider Web Site: http://jakarta.apache.org/commons/

Software Provider: Apache Software Foundation
Software Name: org.apache.commons.collections
Software Version: 2.1
Jar File Name: commons-collections.jar
Provider Web Site: http://jakarta.apache.org/commons/

Software Provider: Apache Software Foundation
Software Name: Commons Database Connection Pooling
Software Version: 1.1
Jar File Name: commons-dbcp-1.1.jar
Provider Web Site: http://jakarta.apache.org/commons/

Software Provider: Apache Software Foundation
Software Name: Jakarta Commons Digester
Software Version: 1.6
Jar File Name: commons-digester.jar
Provider Web Site: http://jakarta.apache.org/commons/

Software Provider: Apache Software Foundation
Software Name: Jakarta Commons Lang
Software Version: 2.0
Jar File Name: commons-lang.jar
Provider Web Site: http://jakarta.apache.org/commons/

Software Provider: Apache Software Foundation
Software Name: org.apache.commons.logging
Software Version: 1.0.4
Jar File Name: commons-logging.jar
Provider Web Site: http://jakarta.apache.org/commons/

Software Provider: Apache Software Foundation
Software Name: Commons Object Pooling Library
Software Version: 1.1
Jar File Name: commons-pool-1.1.jar
Provider Web Site: http://jakarta.apache.org/commons/

Software Provider: Apache Software Foundation
Software Name: Commons Validator
Software Version: 1.1.3
Jar File Name: commons-validator-1.1.3.jar
Provider Web Site: http://jakarta.apache.org/commons/

Software Provider: METASTUFF, LTD.
Software Name: dom4j
Software Version: 1.3
Jar File Name: dom4j.jar
Provider Web Site: http://www.dom4j.org/

Software Provider: Apache Software Foundation
Software Name: Ehcache
Software Version: 1.1
Jar File Name: ehcache-1.1.jar
Provider Web Site: http://ehcache.sourceforge.net/

Software Provider: JGoodies
Software Name: JGoodies Forms
Software Version: 1.0.3
Jar File Name: forms-1.0.3.jar
Provider Web Site: http://www.jgoodies.com/freeware/forms/

Software Provider: Hibernate
Software Name: Hibernate
Software Version: 2.1.7
Jar File Name: hibernate2.jar
Provider Web Site: http://www.hibernate.org/

Software Provider: IBM
Software Name: com.ibm.jvm.classloader
Software Version:
Jar File Name: ibmext.jar
Provider Web Site: www.ibm.com

Software Provider: Apache Software Foundation
Software Name: Jakarta Regexp
Software Version: n/a
Jar File Name: jakarta-regexp.jar
Provider Web Site: http://jakarta.apache.org/regexp/

Software Provider: JDOM.org
Software Name: JDOM
Software Version: 1.0beta9
Jar File Name: jdom.jar
Provider Web Site: http://www.jdom.org/

Software Provider: ObjectWeb
Software Name: JOTM
Software Version: 1.4.1
Jar File Name: jotm.jar
Provider Web Site: http://www.objectweb.org/

Software Provider: Apache Software Foundation
Software Name: org/apache/log4j/
Software Version: 1.2.13
Jar File Name: log4j.jar
Provider Web Site: http://logging.apache.org/log4j/docs/

Software Provider: JGoodies Karsten Lentzsch
Software Name: JGoodies Looks
Software Version: 1.1.3
Jar File Name: looks-1.1.3.jar
Provider Web Site: http://www.jgoodies.com/freeware/looks/index.html

Software Provider: P6Spy
Software Name: P6Spy
Software Version: n/a
Jar File Name: p6spy.jar
Provider Web Site: http://www.p6spy.com/

Software Provider: Manfred Duchrow Consulting & Software
Software Name: Programmer's Friend Java Libraries
Software Version: 2.2
Jar File Name: pf.jar
Provider Web Site: http://www.programmers-friend.org/

Software Provider: Manfred Duchrow Consulting & Software
Software Name: Programmer's Friend Java Object Inspector
Software Version: 2.0
Jar File Name: pf-joi-full.jar
Provider Web Site: http://www.programmers-friend.org/

Software Provider: IBM Corp.
Software Name: adapter
Software Version: 5.0
Jar File Name: rsadapterspi.jar
Provider Web Site: www.ibm.com

Software Provider: IBM Corp.
Software Name: adapter
Software Version: 5.0
Jar File Name: rsaexternal.jar
Provider Web Site: www.ibm.com

Software Provider: Apache Software Foundation
Software Name: Apache Xerces
Software Version: n/a
Jar File Name: xerces.jar
Provider Web Site: http://xerces.apache.org/

1
Introduction

Overview—What Is RPM?

RPM is a pricing and promotions execution system. Functionality includes the definition, maintenance, and review of price changes, clearances and promotions. Capabilities range from simple item price changes at a single location to complex promotions across zones.

RPM contains three primary pricing execution dialogs for creating and maintaining regular price changes, clearances, and promotions. Although each of the three pricing activities is unique, the system displays these dialogs using a common look and feel. Each of these dialogs uses the conflict checking engine which leverages the RPM future retail table.

The future retail table provides a forward-looking view of all pending approved pricing events affecting an item at a given location.

RPM pricing events are defined against the zone structure. The zone structure represents groups of locations organized to support a retailers pricing strategy. RPM allows the user to break out of the zone structure and create location level events as needed.

RPM supports the definition and application of price guides to these pricing events. Price guides allow the retailer to smooth retails and provide ends in logic to derive a final consumer price.

The system also supports area differential pricing strategies for regular retail price changes. This functionality allows a retailer to define pricing relationships that ease pricing maintenance across the organization.

2

Backend System Administration and Configuration

This chapter of the operations guide is intended for administrators who provide support and monitor the running system.

The content in this chapter is not procedural; rather, it is meant to provide descriptive overviews of the key system parameters.

Supported Environments

See the RPM Installation Guide for information about requirements for the following:

- RDBMS operating system
- RDBMS version
- Middle tier server operating system
- Middle tier
- Compiler

Exception Handling

The two primary types of exceptions within the RPM system are the following:

- System exceptions
  For example, server connection and/or database issues are system exceptions. System exceptions, such as when the server is lost, can bring the system to a halt.
- Business exceptions
  This exception indicates that a business rule has been violated. Most exceptions that arise in the system are business exceptions, a user tries to approve a price change that causes a negative retail.

Configuration Files

Key system configuration parameters are described in this section. When retailers install RPM into an environment, they must update these values to their specific settings. In general, the parameters that do not need to be changed by the client are not mentioned in this section.

rpm.jnlp

The Java Network Launching Protocol (JNLP) launch file is an XML document for Java Web Start. This file describes various locations of code and dynamically downloads updates. This file includes the web server information that is hosting port(s). This file also includes the RMI port on which the application server communicates. For example, if a retailer were to change a host name or change the port that the web server is running on, the retailer would make applicable changes to this file.

Data Source Configuration in Container

Data source settings for the RPM application in AS10g are kept inside of the ear file deployment. The RPM application installer configures all necessary settings for the data source. To change the data source settings for the RPM application after it has been deployed, the following steps must be completed:

1. Log into the Enterprise Manager web interface.
2. Navigate to the OC4J instance running the RPM application.
3. Click the *Administration* tab.
4. Under *Services*, click *JDBC Resources*.
5. Under *Connection Pools*, click RPM Connection Pool.
6. Make necessary changes to the JDBC URL, username, and password,
7. Click Apply.
8. If the schema owner is changed, also make this change in the rpm.properties file in the deployment. Log into the UNIX server and change directories to <ORACLE_HOME>/j2ee/<rpm_oc4j_instance>/applications/<rpm_app_name>/conf and modify the rpm.properties file with the new schema_owner value. This value **must** be in all capital letters. Save the file.
9. Restart the OC4J instance running RPM.

rib_user.properties

There must be a rib_user.properties file located in conf/retek. This properties file is used to log on to the system at the beginning of each injector. Any data changes that happen as a result of the RIB has this user listed if users are tracked with regard to create/update/approve actions. The file is populated with the values below. For more information about the RIB, see "Chapter 4 – Integration Methods and Communication Flow" and RIB documentation.

- rib.user=valid user for the system
- rib.password=password for the above user

Configuration for Oracle Retail Service Layer (RSL) with services_rpm.xml

The RPM service factory configuration file (services_rpm.xml) specifies the mapping between the RPM application and its application services interfaces and their associated implementations. Within this file are flavorsets, which are used to configure the ServiceFactory. RSL requires a flavorset of businesslogic, which is used to distinguish the correct implementation of business logic to use for RPM. For more information about RSL, see "Chapter 4 – Integration Methods and Communication Flow."

For example:

```
retek/services_rpm.xml:
<?xml version="1.0" encoding="UTF-8"?>
<services-config>
    <customizations>
        <interface package="com.retek.rsl.rpm">
            <impl package="com.retek.rpm.app.core.service" />
        </interface>
    </customizations>
</services-config>

retek/service_flavors.xml:
<?xml version="1.0" encoding="UTF-8"?>
<services-config>
    <flavors set="businesslogic">
        <flavor name="java"
locator="com.retek.platform.service.SimpleServiceLocator" suffix="Java"/>
    </flavors>
</services-config>
```

RSM-related Configuration File

A configuration file called jndi_providers_rsm.xml is an RPM-configured file that must be present so that the application can communicate with RSM services and other underlying architecture components for authentication and authorization purposes. Beyond installation, a retailer does not have to change the settings in this file. For a general description of RSM, see "Chapter 3 – Technical Architecture" and RSM documentation.

Logging

Jakarta Commons Logging

The API that RPM components work with is built using Jakarta's Commons Logging package. Commons logging provides “an ultra-thin bridge between different logging libraries”, enabling the RPM application to remain reasonably ”pluggable” with respect to different logger implementations. Objects in RPM that require logging functionality maintain a handle to a Log object, which adapts logging requests to the (runtime configurable) logging provider.

In RPM, Log4j is the library under commons logging.

Additional information about Jakarta Commons Logging can be found at the following websites:

- http://jakarta.apache.org/commons/logging/
- http://jakarta.apache.org/commons/logging/api/index.html

Log4j.xml

The logging mechanism that is used for RPM is log4j.xml, which is the same as the flat text log file for the server. This logging mechanism reveals errors and other significant events that occur during system runtime processing. In most cases, business exceptions and system exceptions rise to the user interface. If an exception is displayed, it is logged. Log4j.xml is an open source product.

Significant application server logging occurs in RPM that should also be configured and monitored. See applicable application server documentation for more information.

Additional information about log4j can be found at the following website:

- http://jakarta.apache.org/log4j/docs/index.html

log4j.xml for RPM is found in the following location:
<ORACLE_HOME>/j2ee/<rpm_oc4j_instance>/applications/<rpm_app_name>/conf/

Logging Levels

The level setting established in log4j.xml instructs the system to log that level of error and errors above that level. The logging levels are the following:

- Fatal
- Error
- Warning
- Info
- Debug

> **Note:** In a production environment, the logging setting should be set to Error or Warn, so that system performance is not adversely impacted.

The level is established in the log4j.xml file.

For example:

```
<!-- ======================= -->
  <!-- Setup the loggers       -->
  <!-- ======================= -->

  <logger name="com.retek">
    <level value="ERROR"/>
  </logger>
```

Output Files

RPM logging output is sent to the console and written to files by the application server. In a default AS10g configuration, the output is written to a file of the format OC4J~<rpm_oc4j_instance>~default_group~1 under <ORACLE_HOME>/opmn/logs. The OC4J instance can be configured to write logs to a different location (for example, <ORACLE_HOME>/j2ee/<rpm_oc4j_instance>/log), and to roll them according to file size. For instructions related to this procedure, see the OC4J Configuration and Administration Guide.

Hibernate Logging

The Hibernate internal logging setting is established in log4j.xml. The commons-logging service directs output to log4j. To use log4j, the log4j.properties file must be in the classpath. An example properties file is distributed with Hibernate. The class to be logged and the logging level can be specified. For a general description of Hibernate, see "Chapter 3 – Technical Architecture".

For example:

```
!-- ======================================================== -->
  <!-- Hibernate trace at this level to log SQL parameters    -->
  <!-- ======================================================== -->

  <logger name="net.sf.hibernate.engine.QueryParameters">
    <level value="TRACE"/>
  </logger>
```

Transaction Timeout and Client Inactivity Timeout

This section describes how to establish settings for a transaction timeout. A transaction timeout is the maximum duration, in seconds, for transactions on the application server. Any transaction that is not requested to complete before this timeout is rolled back.

To set up these timeouts, complete these steps:

1. Log into the Enterprise Manager web interface and then navigate to the OC4J instance running the RPM application.
2. Click on the *Administration* tab.
3. Under *Services*, click the *Transaction Manager (JTA)*.
4. Click the *Administration* tab.
5. Under *General*, set the *Transaction Timeout* setting (for example, 600 seconds).

RPMTaskMDB

RPMTaskMDB is a message driven bean used to facilitate RPM 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.

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

EJBs Used by RPMTaskMDB

com.retek.rpm.app.task.service.RPMTaskAppServiceBmtEjb

Tables Used by RPMTaskMDB

- RPM_TASK, RPM_CONFLICT_CHECK_TASK, RPM_LOCATION_MOVE_TASK

  Current, past, and pending tasks to be executed.
- ALERTS, ALERT_RECEIVER, ALERT_STATUS, ALERT_STATUS_DSC

  Tables used for sending alerts to users about task status.

Simplified RPM

For clients who want a less complex version of RPM and the lower costs of ownership associated with it, a simplified version of RPM is available.

The simplified version of RPM is limited only by the security settings in RSM (which is established at the time of installation).

Simplified RPM is configured through RSM seed scripts. Running the simplified version of the RSM seed scripts populates the named_permission and named_permission_dsc tables with only the tasks that are available for Simplified RPM. Only those tasks are then available within RSM when assigning task permissions to roles. The configuration of the security settings is determined by install scripts that are available with the RSM installation:

- RSM_RPM_SE_named_permission.SQL

  This script defines the named permissions and actions associated with them for RSM. This script should be run when simplified RPM is installed.
- RSM_RPM_named_permission.SQL

  This script defines the named permissions and actions associated with them for RSM. This script should be run in addition to the RSM_RPM_SE_named_permission.SQL script when a client is using enterprise RPM.
- RSM_RPM_SE_named_permission_dsc.SQL

  This set of scripts defines the named permission descriptions and the language settings for the named permissions. These scripts should be run when simplified RPM is installed. There is one script for each supported language. They are named "RSM_RPM_SE_named_permission_dsc_*.SQL where * is the language code with an optionial country code.
- RSM_RPM_named_permission_dsc.SQL

  This set of scripts defines the named permission descriptions and the language settings for the named permissions. These scripts should be run in addition to the RSM_RPM_SE_named_permission_dsc.SQL scripts when enterprise RPM is installed. There is one script for each supported language. Each one is named RSM_RPM_named_permission_dsc_*.SQL, where * is the language code with an optional country code.

The functionality of both versions of RPM is outlined below:

- System Options
- Foundation
- Link Codes
- Zone Structure
- Price Guides
- Price Changes
- Clearances
- Promotion Events
- Promotions
- Promotion Threshold
- Promotion Constraints
- Vendor Funding Defaults

- Conflict Checking Results
- Price Inquiry

Enhanced Pricing Functionality

This functionality is NOT available in simplified RPM:

- Pricing Strategies
- Worksheets
- Candidate Rules
- Calendars
- Market Basket Codes
- Aggregation Level

Simplified RPM Batch Program Notes

When Simplified RPM is enabled (RPM Simplified Indicator is enabled) then the following batch programs must be turned off from the integrated batch schedule.

- PriceStrategyCalendarBatch
- WorksheetAutoApproveBatch
- MerchExtractKickOffBatch

Configuring RPM without the RIB

RPM integrates with RMS through the RIB. Because both RPM and RMS reside on the same database, the RIB integration may not be necessary for all clients. Therefore, the client can configure the system to either use the RIB or by-pass the RIB for RMS/RPM integration, depending on the value of the RPM_RIB_IND parameter from the system_options table (N for the No-RIB configuration or Y). See the RMS Installation Guide for additional information on setting the value of the system_options table.

> **Note:** It is assumed that RibForRPM is not installed and does not need to be configured to stop processing incoming RIB messages from RMS.

1. Log into the UNIX server as the user who has write access under ORACLE_HOME.
2. Change directories to <ORACLE_HOME>/opmn/conf.
3. Make a backup of opmn.xml and then edit the file. Locate the process-type element for the OC4J instance running the RPM application.
4. Under the process-type, under start-parameters -> java-options, add the following value:

   ```
   -Dretek.no.rib=true
   ```

   For example:

   ```
   <process-type id="rpm-oc4j-instance" module-id="OC4J" status="enabled">
      <module-data>
         <category id="start-parameters">
            <data id="java-options" value="-Dretek.no.rib=true …
   ```

5. To publish price events using the Publish/Export batches, add the following property/value to rpm.properties on the server:

   ```
   delete_staged_rib_payloads=false
   ```

6. The following triggers must be enabled so that data will be written to the staging table for RMS-RPM integration:
   - RMS_TABLE_RPM_DEP_AIR (on DEPS table)
   - RMS_TABLE_RPM_ITL_AIUDR (on ITEM_LOC table)

   They can be enabled with the following syntax:
   `ALTER TRIGGER [schema.]trigger ENABLE`

No RIB Publishing

RPM has three configuration options regarding RIB publishing:

1. Delete_staged_rib_payloads=true | false (default is true): configured in rpm.properties
2. retek.no.rib=true | false (default is false): configured via JVM system property
3. retek.no.rib.publish=true | false (default is false): configured via JVM system property

This is how RPM will work with each combination of these properties. (The first row is default settings):

Configuration Settings			Results		
delete_staged_rib_payloads	**retek.no.rib**	**retek.no.rib.publish**	**Receive messages from RIB**	**Publish messages to RIB**	**Data remains in staging tables**
TRUE	FALSE	FALSE	Yes	Yes	No
TRUE	FALSE	TRUE	Yes	No	No
TRUE	TRUE	<ANY>	No	No	No
FALSE	FALSE	FALSE	Yes	Yes	Yes
FALSE	FALSE	TRUE	Yes	No	Yes
FALSE	TRUE	<ANY>	No	No	Yes

This is how to set the options for the different results:

- Receive messages from RIB: retek.no.rib=false
- Publish messages to RIB: retek.no.rib=false AND retek.no.rib.publish=false
- Leave data in staging tables: delete_staged_rib_payloads=false

Configurable RIB Batch Program Notes

When the RIB is used for exchanging all messages between RMS and RPM, then the following batch programs must be turned off from the integrated batch schedule.

- NewItemLocBatch
- ItemLocDeleteBatch

Disabling RIB Publishing in RPM

The steps below describe how a retailer can disable RIB publishing in RPM. One reason for this procedure is that a retailer may wish to run a test but not want results published to the RIB. For more information about the RIB, see "Chapter 4 – Integration Methods and Communication Flow" and RIB documentation.

1. Log into the UNIX server as the user who has write access under ORACLE_HOME.
2. Change directories to <ORACLE_HOME>/opmn/conf.
3. Make a backup of opmn.xml and then edit the file. Locate the process-type element for the OC4J instance running the RPM application.
4. Under the process-type, under start-parameters -> java-options, add the following value:

   ```
   -Dretek.no.rib=true
   ```

   For example:

   ```
   <process-type id="rpm-oc4j-instance" module-id="OC4J" status="enabled">
      <module-data>
         <category id="start-parameters">
            <data id="java-options" value="-Dretek.no.rib=true …
   ```

5. Save opmn.xml
6. Reload OPMN and restart the OC4J instance running the RPM application.

   For example:

   ```
   $ORACLE_HOME/opmn/bin/opmnctl reload
   $ORACLE_HOME/opmn/bin/opmnctl restartproc process-type=rpm-oc4j-instance
   ```

Internationalization

Internationalization is the process of preparing software for release into international markets by ensuring it can efficiently handle multiple languages.

This section describes the configuration settings and features within RPM that ensure that the base application can handle multiple languages.

Translation

Translation is the process of interpreting and adapting text from one language into another. Although the code itself is not translated, components of the application that are translated include the following:

- Graphical user interface (GUI)
- Error messages

Supported Languages

RPM supports the following languages in all GA releases:

- Chinese (Traditional)—*_zh_TW.properties
- Chinese (Simplified)—*_zh.properties
- English—(this is the default language)
- French—*_fr.properties
- German—*_de.properties
- Italian—*_it.properties
- Japanese—*_ja.properties
- Korean—*_ko.properties
- Portuguese (Brazilian)—*_pt.properties
- Russian—*_ru.properties
- Spanish—*_es.properties

Set the Client Operating System to the Applicable Locale

For a client machine to leverage internationalization, set the operating system to the appropriate locale. Below is the procedure for setting a Microsoft Windows XP OS to a particular language. For other operating systems, please consult the operating system's guide.

> **Note:** The required language must be installed according to Microsoft instructions before setting regional and language options.

1. From the Control Panel, select Regional and Language Options. The Regional and Language Options window displays.
2. Select the required language from the Standards and formats drop-down field.
3. Click **OK**.

Resources_xx.properties

The text in the .properties files below is translated so that the interface functions in local settings. When a country code other than the default is used, the retailer populates the

_xx.properties files below, where the applicable country equals xx. Much of what is locale-specific in RPM has been pulled out of the code and placed into the following files.

- messages.properties and messages_xx.properties
- resources.properties and resources_xx.properties
- codes.properties and codes_xx.properties
- application_definition_rpm_messages.properties (contains labels that are displayed in the RSM GUI for data security setup)
- worksheet_column_names.properties and worksheet_column_names_xx.properties (contains the labels from the worksheet details table columns)

As shown below, the properties files can be found in rpm_client_properties.jar and rpm_server_properties.jar.

- rpm_client_properties.jar
  - rpm12-ui/src/com/retek/rpm/gui/Resources.properties
  - rpm12-ui/src/com/retek/rpm/gui/worksheet/Worksheet_column_names.properties
- rpm_server_properties.jar
  - rpm12-server/conf/retek/messages.properties
  - rpm12-server/conf/retek/codes.properties
  - rpm12-server/conf/retek/application_definition_rpm_messages.properties
- RPM resides on platform code that has its own resource bundles. They are in the platform-resources.jar file.

Price Management Status Page

Because RPM depends on a number of servers--and because a number of Oracle Retail products are dependent on RPM--a status page helps the retailer determine whether RPM and associated servers are running correctly. Privileges to this page can be set in Oracle Retail Security Manager and are typically reserved for administrators. The status page application displays the answers to the following questions:

- Is the RPM/RMS_Database running?
- Is the RPM JMS Server running?
- Is RSM running?
- Can the application get access to the RPM service?
- Can the application log in to RPM?
- Can RPM data be retrieved?
- Can RMS data be retrieved?
- Is the application able to publish to RIB each message type?

Sample Output

The text below represents sample output, where the questions above have been confirmed as Yes.

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.frecon retek S


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.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 PRMPRCCHG.PRMPRCCHGDEL is ON
************************The above exception indicates that we have passed
**************************
**************************************
Starting Report
RpmJmsServerCheck Passed

Done.
```

3

Technical Architecture

This chapter describes the overall software architecture for RPM. The chapter provides a high-level discussion of the general structure of the system, including the various layers of Java code. From the content, integrators can learn both about the pieces of the system and how they interact.

A description of RPM-related Java terms and standards is provided at the end of this chapter.

Overview

RPM architecture is built on a layered model. That is, layers of the application communicate with one another through an established hierarchy and only with neighboring layers. Any given layer need not be concerned with the internal functional tasks of any other layer.

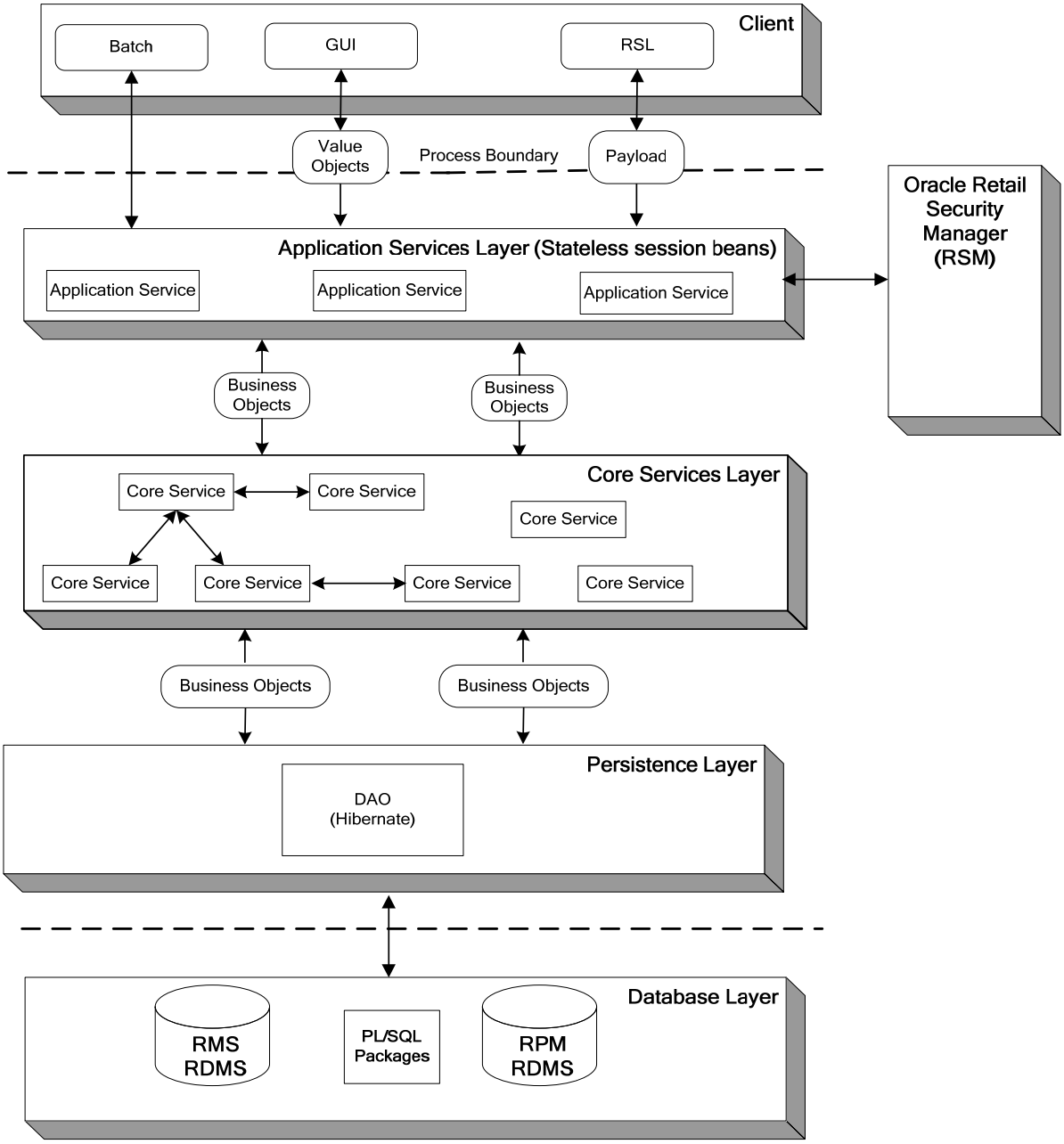
Conceptually, the RPM J2EE is service-oriented architecture built on four levels. In short, it is a collection of "services" that pass data, perform business processing, coordinate system activities, and render data into abstract objects. A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services within the system.

The benefits of layered Java include the following:

- The separation of presentation, business logic, and data makes the software cleaner, more maintainable, and easier to modify.
- The look and feel of the application can be updated more easily, because the GUI is not tightly coupled to the backend.
- Java applications have enhanced portability, which means the application is not locked into a single platform. Upgrades are easier to implement, and hardware is easier to change.
- Logic is implemented using Java objects within a core services layer that is designed around proven architecture concepts.

The Layered Model

The following diagram offers a high-level conceptual view of the service-oriented architecture within RPM. The diagram highlights the separation of layers, as well as their responsibilities within the overall architecture. Key areas of the diagram are described in more detail in the sections that follow.



RPM Technical Architecture

Client

The client layer comprises the GUI and interfaces. The GUI presents data to the user and receives data directly from the user through the front end. The GUI was developed using a Java Swing framework, which is a toolkit for creating rich presentation in Java applications. A design library defines "look and feel".

The Oracle Retail Service Layer (RSL) and batch processing interfaces also operate as clients to the application. They are interface points that interact with the application services layer.

For more information about how RSL integrates with RPM, see "Chapter 4 – Integration Methods and Communication Flow." For more information about batch processing, see "Chapter 6 – Java and RETL Batch Processes."

Application Services Layer (Stateless Session Beans)

Application services meet specific, often narrow, data requirements for the client. They reside on the server side of the process boundary (also known as the remote call boundary).

The application services layer of RPM architecture implements the enterprise Java bean (EJB) type called stateless session beans (SSB). A stateless session bean could be designed for the GUI, for example.

The application-specific services layer provides an interface between a client and the adjacent core services layer. To solve a business problem, application services call one or more core services. Application services also could call other application services

An important way that application services accept incoming data from a client is through value objects and/or payloads. A value object is a data holder in a highly-flat form (similar to a bean). Value objects facilitate improved system performance. For example, from the GUI, the value object data only has to be what is needed by an applicable screen or set of screens. A payload holds the data that satisfies the needs of the applicable interface (RSL, for example).

The primary function of an application services layer is to facilitate the conversion of value objects/payloads to business objects--and business objects to value objects/payloads required by the adjacent layers. The value objects/payloads accepted from and returned to the application services layer are data-centric classes that encapsulate closely-related items. Value objects/payloads are used to provide a quick and lightweight method to transfer flat-data items. The value objects/payloads passed between the application services layer and the application services layer contain very little, if any, data-processing logic and, in the context of the RPM, are used solely to transfer data.

The application services depend on both core services and business objects, translating back and forth between input from the client and business objects in the core services layer. The application services call the applicable core service at the applicable time.

Core Services Layer

This layer consists of a collection of separate and distinct services that encapsulate RPM core business logic. Core services are "core" in that they work with the business object model and contain the business object rules for the application. Unlike application services, core services make no presumptions about how they might be used. In other words, core services contain generic views of business functionality, as opposed to a narrow application service process.

Residing very closely to the core services, business objects represent business problems. Business objects contain behaviors. For example, they perform validation and guard themselves from being used improperly. To ensure the atomicity, consistency, isolation, and durability (ACID) properties of state transitions, RPM implements some business logic using a state machine (a workflow engine). Each object with a lifecycle has a state machine, which describes the lifecycle of the object.

Sometimes core services drive processes with business objects. But more often, core services are responsible for finding the business objects and sending them back to the persistence layer. The core services layer is then responsible for managing object persistence by interacting with the data access objects residing in the supporting persistence layer.

To summarize, the core service layer consists of a collection of Java classes that implement the business related logic of the application through one or more high-level methods. The core services represent all logical tasks that can be performed on business objects.

Persistence Layer

RPM uses Hibernate, an object/relational persistence and query service for Java. This object-relational framework provides the ability to map business objects residing in the core services layer to relational tables contained within the data store.

Conceptually, Hibernate encompasses most of the persistence layer. Hibernate interacts with core services by passing/accepting business objects to/from the core services layer. Internally, Hibernate manages the conversion of RPM business objects to relational data elements required by the supporting relational database management system (RDMS).

For information about Hibernate-related logging, see "Chapter 2 – Backend System Administration and Configuration."

Database Layer

The database tier is the storage platform of the application. It contains the physical data used throughout the application. The system is designed to include two RDMS datasources: RPM and RMS.

Security

The RSM application provides basic authorization and authentication functionality during user logon. To perform authentication, RPM has a set of APIs that calls the API tier within Oracle Retail Security Manager (RSM). Using a remote EJB call, objects in RPM "talk" to objects in RSM using Internet Inter-ORB Protocol (IIOP). See the RSM Operations Guide for information about how it performs authentication using Light Directory Access Protocol (LDAP) and a third-party directory server.

Conflict Check Processing

Conflict checking is done in RPM to ensure that rules are followed 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 synchronous or asynchronous. If synchronous, the process 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, 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:** Selecting any of the above options enables asynchronous conflict checking for that functional area.

Synchronous and Asynchronous Processing

When conflict checking is performed *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 always is done synchronously in the following situations:

- RSL calls
- RIB calls
- Location Move Batch
- 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. An alert is issued when conflict checking is completed.

The Conflict Check Results workspace allows the client 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. See the RPM User Guide for instructions on how to view the results of background conflict checking.

Conflict checking for location move scheduling 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. Accordingly, conflict checking for location move scheduling always is asynchronous.

Constraints are performed asynchronously at the same time as the conflict check. When action is executed that kicks off a conflict check, the client must indicate whether to check constraints. 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 application service layer on the server 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 TASK table. The task-specific is saved 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, which is a message-driven bean used to facilitate 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/failure messages and 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 Cached Objects

RPM can cache objects within the server to reduce repeated creation and loading of frequently-accessed information from the database. This strategy helps in reducing database access latency and bottlenecks. The object caches are accessible across requests and users and are configured to be refreshed at configurable time intervals.

RPM caches certain types of business objects and database query results. The business objects that RPM caches are those that are frequently accessed but infrequently changed. RPM sets a cache for each of the following business objects:

- System Date (VDATE)
- System Options
- System Defaults
- Divisions
- Groups
- Departments
- Districts
- Stores
- Warehouses
- Suppliers
- Partners
- Dynamic Codes
- Units Of Measure
- Zone Groups
- Zones
- Differentiators
- Differentiator Types
- Reference Items
- Class Hierarchies
- Subclasses
- Deal Status Codes
- Deal Type Codes

RPM also caches results from non-parameterized database queries. Non-parameterized database queries usually happen when RPM needs data that do not meet specific criteria. For example, all clearance information for the Clearance Workflow. In contrast, parameterized database queries retrieve data that meet specific criteria, such as all clearances affecting a given location.

RPM uses an open-source product called **EHCache** as a caching framework. **EHCache** features both memory and disk cache storage options and flexible configuration. More information about this framework can be found at http://ehcache.sourceforge.net/.

With the exception of the Virtual System Date, the cache for business objects and database query results are configured as follows:

- RPM can store in the memory cache up to 10,000 instances within a business object cache and 10,000 database query result sets. If there are more than 10,000 elements to be stored, the overflow is stored onto disk.
- A cache is refreshed if not accessed for 3600 seconds.

The Virtual System Date (VDATE) differs in configuration in such a way that there is only one instance of a VDATE within its cache and there is no expiration of that instance.

RPM-Related Java Terms and Standards

RPM is deployed using the J2EE-related technologies, coding standards, and design patterns defined in this section.

ACID

ACID represents the four properties of every transaction:

- Atomicity: Either all of the operations bundled in the transaction are performed successfully, or none of them are performed.
- Consistency: The transaction must leave any and all datastores that are affected by the transaction in a consistent state.
- Isolation: From the perspective of the application, the current transaction is independent (in terms of application logic) from all other transactions running concurrently.
- Durability: The operations of a transaction against a datastore must persist.

Data access object (DAO)

This design pattern isolates data access and persistence logic. The rest of the component can then ignore the persistence details (the database type or version, for example).

Java Development Kit (JDK)

Standard Java development tools from Sun Microsystems.

Enterprise Java Beans (EJB)

EJB technology is from Sun. See http://java.sun.com/products/ejb/. EJB refers to a specification for a server-side component model. RPM uses stateless, session EJBs, which are stateless and clusterable--and offer a remotely accessible entry point to an application server.

Enterprise Java Beans (EJB) container

An EJB container is the physical context in which EJBs exist. A container is a physical entity responsible for managing transactions, connection pooling, clustering, and so on. This container manages the execution of enterprise beans for J2EE applications.

J2EE server

The runtime portion of a J2EE product. A J2EE server provides EJB and Web containers.

The Java 2 Enterprise Edition (J2EE)

The Java standard infrastructure for developing and deploying multi-tier applications. Implementations of J2EE provide enterprise-level infrastructure tools that enable such important features as database access, client-server connectivity, distributed transaction management, and security.

Naming conventions in Java

- Packages: The prefix of a unique package name is always written in all-lowercase letters.
- Classes: These descriptive names are unabbreviated nouns that have both lower and upper case letters. The first letter of each internal word is capitalized.
- Interfaces: These descriptive names are unabbreviated nouns that have both lower and upper case letters. The first letter of each internal word is capitalized.
- Methods: Methods begin with a lowercased verb. The first letter of each internal word is capitalized.

Persistence

The protocol for transferring the state of an entity bean between variables and an underlying database.

Remote interface

The client side interface to a service. This interface defines the server-side methods available in the client tier.

Conflict Checking

The major components of Conflict Checking in RPM will be performed by PL/SQL and will be executed on the database server instead of the application server. This addresses performance concerns around this functionality. A number of PL/SQL package files need to be installed on the database.

4

Integration Methods and Communication Flow

This chapter provides information that addresses how RPM integrates with other systems (including other Oracle Retail systems).

This chapter is divided into the following sections that address RPM methods of integration:

- Functional dataflow
- Communication flow diagram and explanation
- Oracle Retail Integration Bus (RIB)-based integration
- Oracle Retail Service Layer (RSL)-based integration
- Persistence layer integration

Functional Dataflow

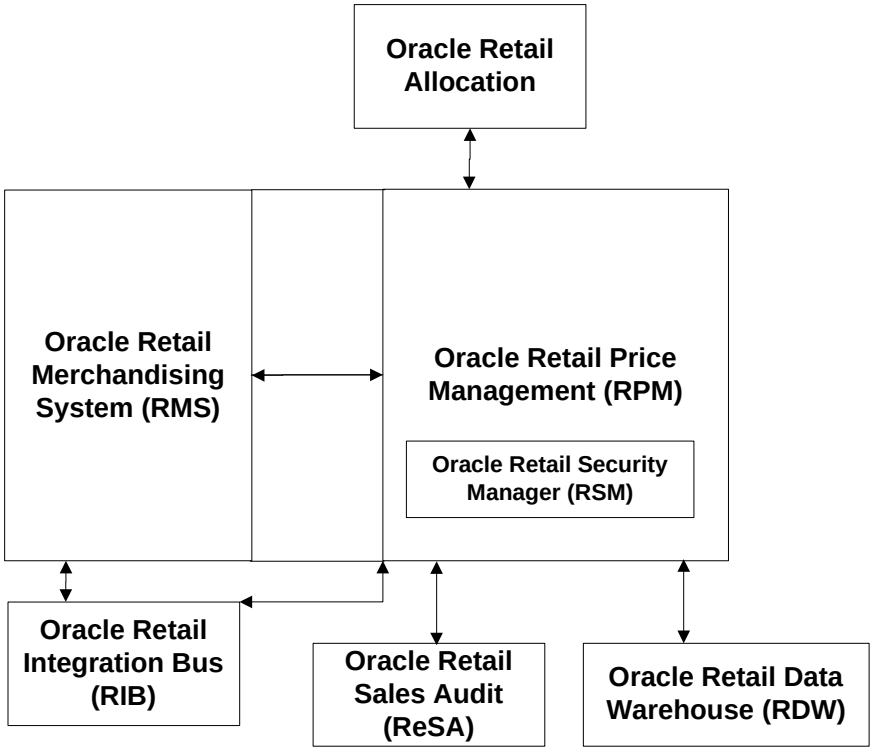
The diagram below details the overall direction of the dataflow among the various systems. The accompanying explanations of this diagram are written from a system-to-system perspective, illustrating the movement of data throughout the RPM-related portion of the enterprise. Note that this discussion focuses on a high-level functional use of data. For a technical description of the dataflow, see the sections later in this chapter.

A Note about the Merchandising System Interface

Many tables and functions within RPM are held in common with the Oracle Retail Merchandising System (RMS). This integration provides the following two important benefits:

- The number of interface points that need to be maintained is minimized.
- The amount of redundant data (required if the rest of the Oracle Retail product suite is installed) is limited.

Integration Interface Dataflow Diagram



RPM-related dataflow across the enterprise

Integration Interface Dataflow Description

From Oracle Retail Allocation to RPM

- Request for future retail price data
  This request is based on information provided by Oracle Retail Allocation (for example, item, location, date).

From RPM to Oracle Retail Allocation

- Future retail price data
  Oracle Retail Allocation uses this data to provide the user with the future retail price value of the entire allocation (based on its quantities). The future retail price data is stored in RPM and consists of approved pricing events (price change, clearance, promotion) that affect an item/location throughout its pricing life. These future retail price values are retrieved by location, date, and item. RPM provides the retail price, the currency, and the price type (regular, clearance, promotion). RPM plans to provide this retail value in the location's currency.

From RPM to RMS

- Price change approval/execution
  RPM publishes price change approvals to RMS so that RMS can generate a ticket request for the specified item/location. RPM also owns price change execution, which is the process of updating the retail information stored in RMS with the new

regular retail prices determined by the regular price change going into effect. RPM processing asks, "Are there any price changes going into effect tomorrow?" If there are, RPM notifies RMS, and RMS updates the retail information with the new regular retail prices. In other words, RMS table updates occur through RMS code.

- Promotion execution
  RPM processing asks, "Are there any promotions (for example, 25% of the retail price of an item) going into effect tomorrow or ending today?" If there are, RPM notifies RMS, and RMS updates the promotional retail information. In other words, RMS table updates occur through RMS code. The promotion of items is frequently driven by a particular event such as a holiday or the overstock of an item. RPM also provides promotion information to RMS so that RMS can associate promotions to orders and/or transfers.
- Clearance execution
  Clearances in RPM are a series of markdowns designed to move inventory out of a store. Clearances always result in the price of an item going down. RPM processing asks, "Are any clearances going into effect tomorrow or resetting tomorrow?" If there are, RPM notifies RMS, and RMS updates the clearance retail information. In other words, RMS table updates occur through RMS code.

  Note: Price changes, clearances, and/or promotions in Deleted status are not applied to item/locations in RMS.

- Initial price data
  Initial retail prices in RMS are derived using various pieces of information stored in RPM. To successfully initially price items in RMS, a primary zone group must be defined in RPM for the merchandise hierarchy assigned to the new item. The primary zone group definition in RPM consists of the following elements:
  - Primary zone group
    The primary zone group determines the structure that is used to initially price the item. When users access the retail by zone link in RMS, they see an initial price for each zone with the primary zone group.
  - Markup percent
    The markup percent is the markup applied to the cost of the item.
  - Markup percent type
    The markup percent type is either cost type or retail type and determines what formula to use when marking up the cost.
  - Price guides
    Pricing guides are used to help create a uniform pricing strategy. They are used to smooth the proposed retails in order to maintain a consistent set of price points.

From RMS to RPM

There are several instances when RMS must notify RPM of actions that occur within RMS. These actions are as follows:

- Store/Warehouse creation
  This notification is used when stores and/or warehouses are added to RMS. RPM needs the new store/warehouse and its associated pricing location to assign the new store/warehouse to the zone structure. The message also contains the currency of the new store/warehouse in the event that the pricing location assigned does not share the same currency as the new store/warehouse.
- Item/Location creation
  This notification is used when a new item/location relationship has been created. RPM processes this message and makes sure that this item/location combination does not have existing future retail records. If the item is sellable, RPM then creates an initial future retail record for this item/location combination along with its retail value. If there are approved promotions/clearances/price changes at the intersection of any level between the merchandise and zone hierarchies, RPM proceeds to attach this new item/location to those price events and, eventually, as new records in the RPM_FUTURE_RETAIL table.
- Item modification
  This notification is used when there is an item reclassification in RMS. RPM uses this information to update the department, class, and subclass information in the RPM_ITEM_MODIFICATION table. When the next scheduled batch process runs, the RPM_FUTURE_RETAIL table is updated with these new values.
- Department creation
  This notification is used when a new department is created in RMS. RPM creates aggregation level information for the new department using predefined system defaults. The user can modify these values through the Maintain Aggregation Levels workflow.

From RPM to RSM

Note: See the RSM Operations Guide for a discussion of of LDAP, named permissions functionality, data permissions functionality, and other RSM operations.

- User and password data that requires authentication
  RPM sends RSM the user and password data that requires authentication. RSM calls the LDAP- compliant directory service to authenticate username and password data. Once a user is authenticated, RSM creates an encrypted user signature (a ticket).

From RSM to RPM

- Encrypted user signature
  The login credentials (user signature) are encrypted because the information is being sent over the wire.
- Names permissions data
  This data maps users to roles, and roles to specific functionality (named permissions).
- Data permissions data
  RSM administers data-level permissions. To facilitate this functionality, any Oracle Retail application utilizing RSM for data level permissions initially populates RSM tables with its hierarchy types (for example, merchandise and location).

From RPM to ReSA

ReSA needs to receive promotion data from RPM, and ReSa retrieves this data through the RETL extract program. Included in this data are promotion detail, promotion events, and promotion headers. For more information regarding this, see "RETL Program Overview for RPM Extractions" in "Chapter 6 – Java and RETL Batch Processes."

From RPM to SIM and from SIM to RPM

RPM publishes information to Oracle Retail Store Inventory Management (SIM) to communicate the status of price changes, clearances, and promotions within the application. The messages are published at a transaction item/location level and include the following price change events:

- Price changes
  - RPM publishes approved price changes at the location level and they are published as "fixed price" price changes, with the price change type and the price guides already applied.
  - RPM publishes price changes that were once approved (published) but are now cancelled/deleted.
  - SIM requests new price changes. RPM checks the submitted price change for conflicts. If the conflict checking is successful, RPM assigns a reason code and price change ID and publishes the approved results. If the conflict checking is unsuccessful, RPM informs SIM of the failure.
  - SIM edits existing price changes. RPM validates the edits to ensure that they do not create conflicts or a negative retail. If conflict checking is successful, approved updates are published. If conflict checking is unsuccessful, RPM publishes a status record relating that the price change was unable to be updated.
- Clearances
  - RPM publishes approved clearance price changes at the location level and they are published as fixed price clearances with the change type and price guides already applied. Clearance changes include a markdown number.
  - RPM publishes the reset when a clearance price changes with a reset date is approved (and therefore the reset date record created).
  - SIM requests a new clearance price change. RPM checks the submitted clearance for conflicts and, if successful, assigns a sequence, a reason code and an ID--and then publishes the approved result. If the conflict checking is unsuccessful, RPM informs SIM of the failure.

  - SIM edits existing clearance price changes. RPM validates the edits to ensure that they do not create conflicts, negative retails, or clearance price raises above the previous clearance retail. Approved updates are published, and SIM is able to implement the clearance. If RPM validation is unsuccessful, RPM informs SIM of the failure.
  - RPM publishes clearances that were once approved (published) but are now cancelled/deleted.
- Promotions
  - RPM publishes approved promotions, including the promotion details. Simple promotions are published with the promotional retail and change type so SIM can apply them to the current regular price or clearance based on the promotion settings. Complex promotions are published with their details, as a specific promotional retail cannot be calculated.
  - SIM creates simple promotions at the item/location level. RPM checks the submitted promotion for conflicts and overlaps. RPM also checks for negative retails to insure that the promo retail is not above the regular retail. Approved promotions are published, and SIM is able to implement the promotion. If RPM validation is unsuccessful, RPM informs SIM of the failure.
  - SIM edits existing simple promotions. RPM validates the edits to ensure they do not create conflicts, negative retails or promotional retails above the regular retail. Approved changes to promotions are published, and SIM is able to implement the changes to the promotion. If RPM validation is unsuccessful, RPM informs SIM of the failure.
  - RPM publishes promotions that were once approved (published) but are now cancelled/deleted.

From RPM to the RIB and from the RIB to RPM

RPM publishes information to the RIB to communicate the status of price changes, clearances, and promotions within the application. The messages are published at a transaction item/location level and include the following price change events:

- Price changes
  - RPM publishes approved price changes at the location level and they are published as “fixed price” price changes, with the price change type and the price guides already applied.
  - RPM publishes price changes that were once approved (published) but are now cancelled/deleted.
  - Clearances
  - RPM publishes approved clearance price changes at the location level and they are published as fixed price clearances with the change type and price guides already applied. Clearance changes include a markdown number.
  - RPM publishes the reset when a clearance price changes with a reset date is approved (and therefore the reset date record created).
  - RPM publishes clearances that were once approved (published) but are now cancelled/deleted.
- Promotions
  - RPM publishes approved promotions, including the promotion details. Simple promotions are published with the promotional retail and change type so that another application can apply them to the current regular price or clearance

based on the promotion settings. Complex promotions are published with their details, as a specific promotional retail cannot be calculated.

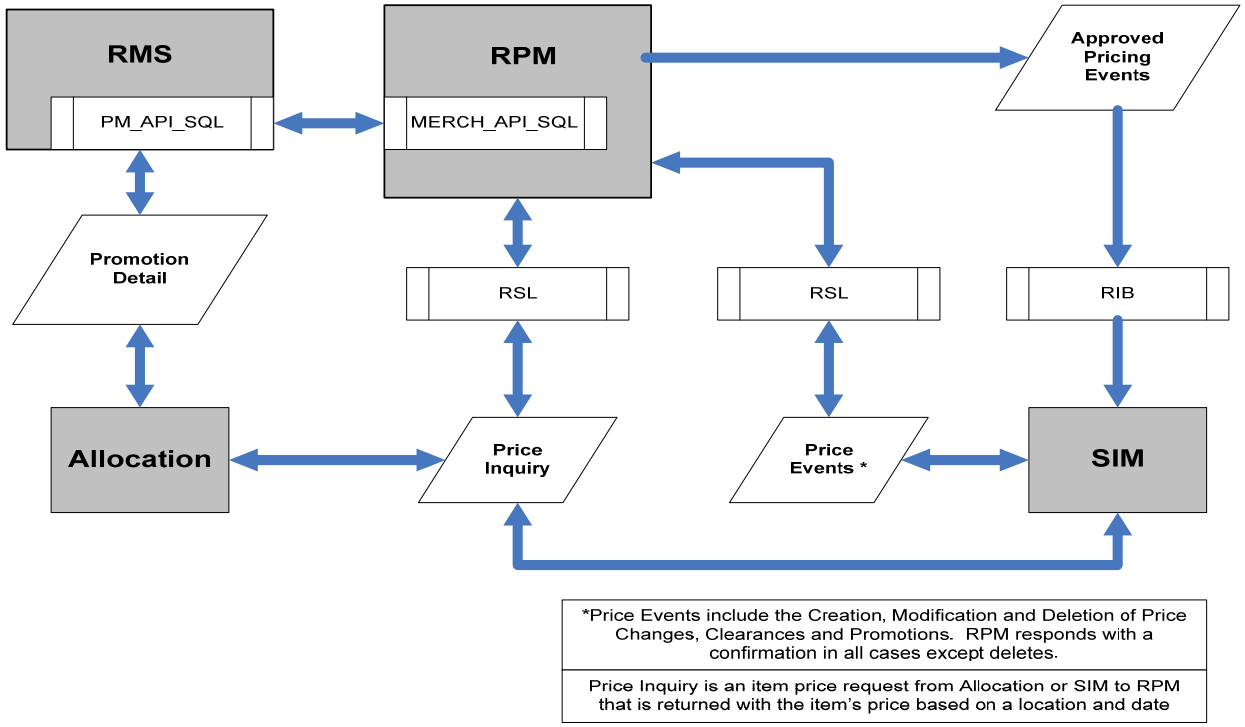
- RPM publishes promotions that were once approved (published) but are now cancelled/deleted.

From RPM to RDW

RDW needs to receive promotion data from RPM, and RDW retrieves this data via the RETL extract program. Included in this data is promotion detail, promotion events, and promotion headers. For more information regarding this, see "RETL Program Overview for RPM Extractions" in "Chapter 6 – Java and RETL Batch Processes."

Pricing Communication Flow Diagram

Pricing detail is communicated from RPM to other applications by various means. The four primary communication components of pricing information are described in this section. A functional explanation of the data follows the diagram.



Pricing communication flow diagram

Approved Price Events

When a price change, clearance or promotion is approved in RPM, the system publishes those events to the Oracle Retail Integration Bus (RIB). Another application can subscribe to the message in order to pass the pricing event information on to the RIB. RPM also publishes messages when approved events are unapproved. This message informs the other application that the event previously sent will not take place.

Price Events

SIM has the ability to create, modify, or delete price changes, clearances and promotions on a store by store basis. SIM communicates these requests via the Oracle Retail Service Layer (RSL) and RPM runs the requests through conflict checking. If the requests pass conflict checking then they become approved price events. RPM sends a confirmation back to SIM for creation and modification requests, but not deletions. As stated above, the approved price events and details are then communicated via the RIB.

Price Inquiry

Oracle Retail Allocation must know the price of an item is on a given day for a given location. Oracle Retail Allocation usually requests a future date. The requests are communicated to RPM via RSL and processed by RPM from the future retail table. RPM sends back the price information for the requested item/location/date combination.

Promotion Detail

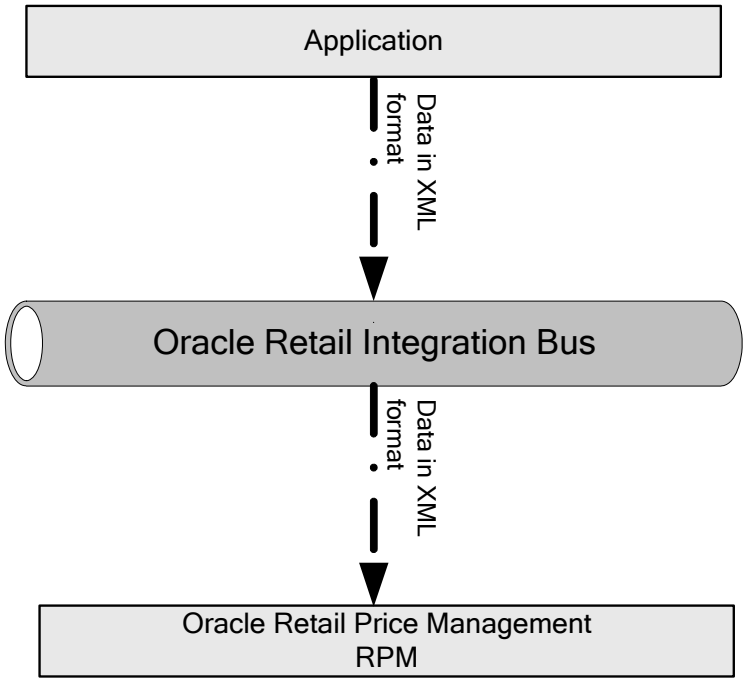
When Oracle Retail Allocation requires promotion detail, it is able to retrieve the data via RMS from RPM. There are two direct package calls involved: Oracle Retail Allocation calling the RMS package, and the RMS package calling an RPM package. This provides Oracle Retail Allocation with the promotional detail beyond what would be provided in a price inquiry request.

RPM and the Oracle Retail Integration Bus (RIB)

The flow diagrams and explanations in this section provide a brief overview of publication and subscription processing. See the latest Oracle Retail Integration documentation for additional information. For information about RIB-related configuration within the RPM application, see the sections ″rib_user.properties″ and ″Disabling RIB Publishing in RPM″ in “Chapter 2 – Backend System Administration and Configuration”.

The XML Message Format

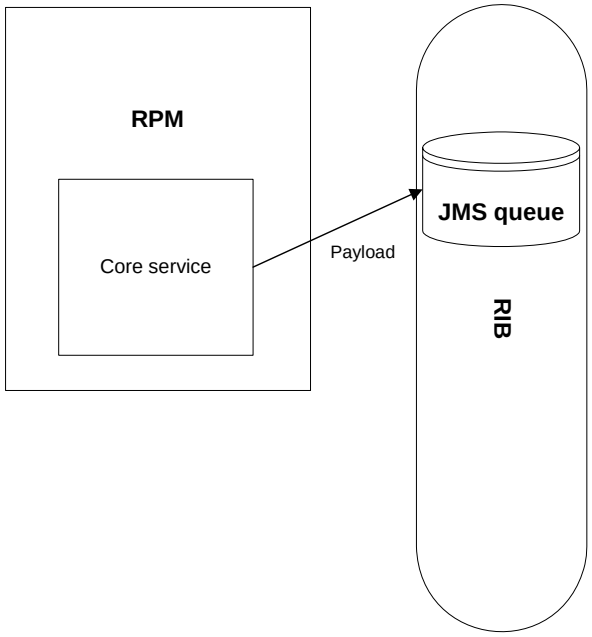
As shown by the diagram below, the messages to which RPM subscribes and which RPM publishes are in an XML format and have their data structure defined by document type definitions (DTDs) or XML schema documents.



Data across the RIB in XML format

Message Publication Processing

As shown by the diagram below, an event within the RPM core service layer (that is, an insert, update, or delete) leads it to write out a payload that is published to the RIB. The RIB engages in polling the JMS queue, searching for the existence of a message. A publishable message that appears on the queue is processed. The RIB is unconcerned about how RPM gets its message to the JMS queue table.

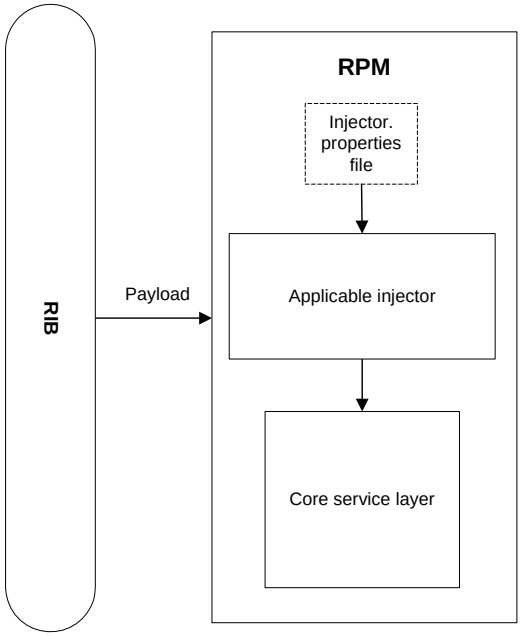


RPM message publication processing

Message Subscription Processing

As shown by the diagram below, based on the message family and the message type, an injector.properties file within RPM knows which injector to instantiate. Note that one injector can handle multiple .dtd messages. The injector "injects" the data into the core service layer, which is configured to act upon and/or validate the data.

Note: Under ordinary runtime conditions, the injector-related properties files shown in the diagram do not have to be modified.



RPM message subscription processing

Publishers Mapping Table

This table illustrates the relationships among the message family, message type and DTD/payload. For more information, see the latest Oracle Retail Integration documentation.

Family	Type	DTD/Payload
regprcchg	regprcchgcre	RegPrcChgDtl
regprcchg	regprcchgmod	RegPrcChgDtl
regprcchg	regprcchgdel	RegPrcChgDtlRef
clrprcchg	clrprcchgcre	ClrPrcChgDtl
clrprcchg	clrprcchgmod	ClrPrcChgDtl
clrprcchg	clrprcchgdel	ClrPrcChgDtlRef
prmprcchg	prmprcchgcre	PrmPrcChgDtl
prmprcchg	prmprcchgmod	PrmPrcChgDtl
prmprcchg	prmprcchgdel	PrmPrcChgDtlRef

Subscribers Mapping Table

The following table lists the message family and message type name, the document type definition (DTD) that describes the XML message, and the subscribing classes that facilitate the entry of the data into core service layer. These classes are described in the code as ”injectors”. For more information, see the latest Oracle Retail Integration documentation.

Family	Type	DTD/Payload	Injector (subscribing class)
store	storecre	StoreDesc	NewLocationInjector
wh	whcre	WHDesc	NewLocationInjector
itemloc	itemloccre	ItemLocDesc	NewItemLocInjector
merchHier	deptcre	MrchHrDeptDesc	NewDepartmentInjector
items	itemhdrmod	ItemHdrDesc	ItemModificationInjector

Functional Descriptions of Messages

The table below briefly describes the functional role that messages play with regard to RPM functionality. The table also illustrates whether RPM is publishing the message to the RIB or subscribing to the message from the RIB, and the Oracle Retail products that are involved with the RIB integration. For more information, see the latest RIB documentation.

Functional Area	Subscription/ Publication	Integration to products	Description
Store Creation	Subscribe	RMS	This message is used to notify RPM when stores are added to RMS. RPM needs the new store and its associated pricing location in order to assign the new store to the zone structure. The message also contains the currency of the new store in the event that the pricing location assigned does not share the same currency as the new store.
Warehouse Creation	Subscribe	RMS	This message is used to notify RPM when warehouses are added to RMS. RPM needs the new warehouse and its associated pricing location in order to assign the new warehouse to the zone structure. The message also contains the currency of the new warehouse in the event that the pricing location assigned does not share the same currency as the new warehouse.

Functional Area	Subscription/ Publication	Integration to products	Description
Item/Location Creation	Subscribe	RMS	This message is used to notify RPM when a new item/location relationship has been created. RPM processes this message and makes sure that this item/location combination does not have existing future retail records. If the item is sellable, RPM then creates an initial future retail record for this item/location combination along with its retail value. If there are approved promotions/clearances/price changes at the intersection of any level between the merchandise and zone hierarchies, RPM proceeds to attach this new item/location to those price events, and, eventually, insert new records in the RPM_FUTURE_RETAIL table.
Item Modification	Subscribe	RMS	This message is used to notify RPM when there is an item reclassification in RMS. RPM uses this information to update the department, class, and subclass information in the RPM_ITEM_MODIFICATION table. When the next scheduled batch process runs, the RPM_FUTURE_RETAIL table is updated with these new values.
Department Creation	Subscribe	RMS	This message is used to notify RPM when a new department is created in RMS. RPM creates and sets default aggregation level data for the new department when the message is processed.
Price Change Creation	Publish	RMS, SIM	This message is used by RPM to communicate the approval of a price change within the application. This message is published at a transaction item/location level.
Price Change Modification	Publish	RMS, SIM	This message is used by RPM to communicate the modification of a new retail on an already approved price change. This message is published at a transaction item/location level.

Functional Area	Subscription/ Publication	Integration to products	Description
Price Change Deletion	Publish	RMS, SIM	This message is used by RPM to communicate the deletion (un-approval included) of an already approved price change. This message is published at a transaction item/location level.
Clearance Creation	Publish	SIM	These messages are used by RPM to communicate the approval of a clearance price change within the application. This message is published at a transaction item/location level, and is used by SIM for visibility to the new clearance retail on the effective date for the clearance price change.
Clearance Modification	Publish	SIM	This message is used by RPM to communicate the approval of a new retail on an already approved clearance price change. This message is published at a transaction item/location level. It is used by SIM for visibility to the modified clearance retail on the effective date for the clearance price change.
Clearance Deletion	Publish	SIM	This message is used by RPM to communicate the deletion (un-approval included) of an already approved clearance price change. This message is published at a transaction level/location level, and is used to notify SIM of the deletion of the clearance price change.
Promotion Creation	Publish	SIM	These messages are used by RPM to communicate the approval of a promotion within the application. This message is published at a transaction item/location level, and is used by SIM for visibility to the new promotional retail on the effective date for the promotion.
Promotion Modification	Publish	SIM	This message is used by RPM to communicate the modification of a new retail on an already approved promotion. This message is at a transaction item/location level. It is used by SIM for visibility to the modified promotional retail on the effective date for the the promotion.

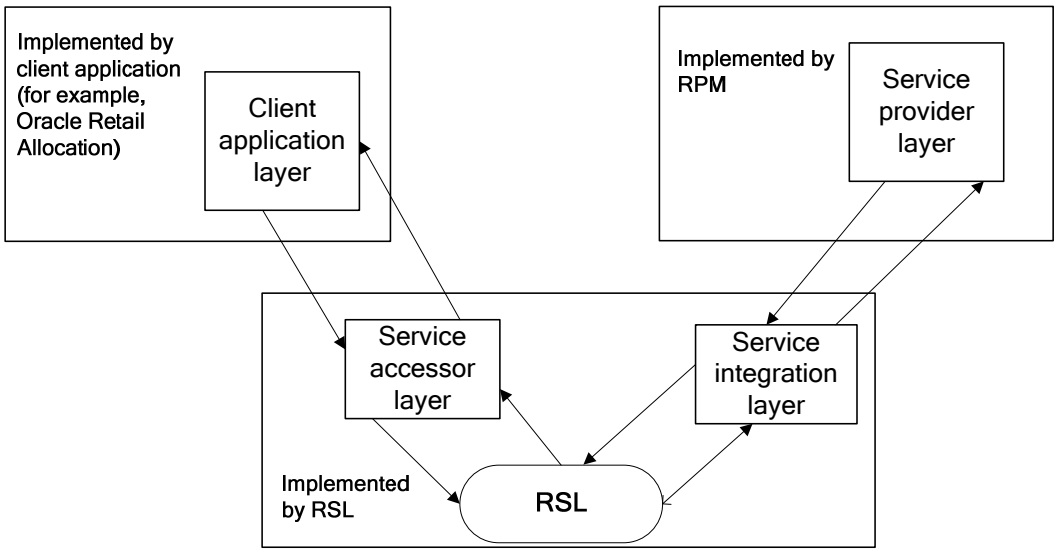
Functional Area	Subscription/ Publication	Integration to products	Description
Promotion Deletion	Publish	SIM	This message is used by RPM to communicate the deletion (un-approval included) of an already approved promotion. This message is published at a transaction item/location level, and is used to notify SIM of the deletion of a promotion.

RPM and the Oracle Retail Service Layer (RSL)

RSL is a framework that allows Oracle Retail applications to expose APIs to other Oracle Retail applications. As shown in the diagram below, in RSL terms, there is a client application layer and a service provider layer. RPM includes the service provider layer that owns the business logic.

The RPM implementation of RSL exposes a *synchronous* method to communicate with other applications. (RIB-facilitated processing is asynchronous.) All RSL services are contained within an interface offered by a Stateless Session Bean (SSB). To a client application, each service appears to be merely a method call.

For information about RSL-related configuration within the RPM application, see "Chapter 2 – Backend System Administration and Configuration."



Client application and service provider processing through RSL

Functional Description of the Class Using RSL

The table below briefly describes the functional role that the RSL class has within the RPM.

Class	Description
PriceInquiryServiceJava.java	This service, provided by RPM, allows an inquiring system to request the effective retail for an item at a specified location on a given date. RPM provides the retail value and indicates whether the value is promotional, clearance or regular.

Persistence Layer Integration

The system is designed to include two RDMS datasources, RPM and RMS. RPM and RMS share certain database tables and processing logic. RPM exchanges data and processing with RMS in four ways:

- By reading directly from RMS tables.
- By directly calling RMS packages.
- By reading RPM views based on RMS tables.
- RMS utilizes RPM packages in order to access processing and information available only in RPM. This type of interaction is only done through package calls.

For more information about RPM's persistence layer and database layer, see "Chapter 3 – Technical Architecture."

RMS Tables Accessed through the Persistence Layer

RPM uses the tables shown below through the persistence layer:

RMS tables accessed through the persistence layer
AREA
CHAIN
CLASS
CODE_DETAIL
CODE_HEAD
COMP_PRICE_HIST
COMP_STORE
COMPETITOR
DEAL_COMP_PROM
DEAL_DETAIL
DEAL_HEAD
DEAL_ITEMLOC
DEPS
DIFF_GROUP_DETAIL
DIFF_GROUP_HEAD
DIFF_IDS
DIFF_TYPE
DISTRICT
DIVISION
FUTURE_COST
GROUPS
ITEM_LOC
ITEM_MASTER

RMS tables accessed through the persistence layer
ITEM_SEASONS
ITEM_SUPPLIER
LOC_LIST_DETAIL
LOC_LIST_HEAD
PARTNER
PHASES
REGION
SEASONS
SKULIST_DETAIL
SKULIST_HEAD
STORE
SUBCLASS
SUPS
SYSTEM_OPTIONS
UDA
UDA_ITEM_FF
UDA_ITEM_LOV
UDA_ITEMDATE
UDA_VALUES
UOM_CLASS
WH

RMS Packages and Methods Accessed through the RPM Persistence Layer

RPM uses the packages and methods shown in the table below through the persistence layer:

RMS packages	RMS methods
RPM_WRAPPER	uom_convert_value
	valid_uom_for_items
	get_vat_rate_include_ind
	currency_convert_value
PM_DEALS_API_SQL	create_deal
	new_deal_comp
RMSSUB_PRICECHANGE	get_price_change

RPM Views Based on RMS Tables

RPM Views	RMS Tables
rpm_item_diff	ITEM_MASTER
	DIFF_GROUP_DETAIL
	DIFF_GROUP_HEAD
rpm_deal_head	DEAL_HEAD
rpm_primary_ref_item	ITEM_MASTER
rpm_future_cost	FUTURE_COST
	SUPS
	ITEM_LOC
rpm_rms_system_options	SYSTEM_OPTIONS
rpm_uda_view	UDA
	UDA_ITEM_DATE
	UDA_ITEM_FF
	UDA_ITEM_LOV

RPM Packages Called by RMS

Packages
MERCH_API_SQL
MERCH_DEALS_API_SQL
MERCH_RETAIL_API_SQL

Oracle Retail Strategic Store Solutions – RPM Integration

Integration between RPM and Oracle Retail Strategic Store Solutions is optional. Clients who are not integrating RPM with Oracle Retail Strategic Store Solutions may skip this section.

Overview

Oracle Retail Strategic Store Solutions Overview

RPM integrates with Oracle Retail Strategic Store Solutions. Applications within Oracle Retail Strategic Store Solutions include the following and more:

- Oracle Retail Point-of-Service (ORPOS)
- Oracle Retail Back Office (ORBO)
- Oracle Retail Central Office (ORCO)

Integration Overview

This section provides an overview as to how RPM is integrated with Oracle Retail Strategic Store Solutions.

A diagram shows the overall direction of the data among the products. The accompanying explanations of this diagram are written from a system-to-system perspective, illustrating the movement of data. For additional information on RMS, ReSA, and RPM integration with Oracle Retail Strategic Store Solutions, see the *Oracle Retail Strategic Store Solutions Implementation Guide*.

RPM sends store-specific information to the Oracle Retail Back Office (ORBO) application. To integrate with ORBO, the RPM extract data output format matches the format that ORBO recognizes. RPM sends three different store specific XML record types:

- Price Change includes both clearance and regular price changes
- Price Promotion includes simple promotions
- Discount Rules includes Threshold and Buy/Get promotions

RPM uses the RPMtoORPOSPublishBatch program to format and stage output of different price events and the RPMtoORPOSPublishExport shell script to produce an xml file based on the output of the RPMtoORPOSPublishBatch. See the Java and RETL Batch Processes chapter for additional details on the batch processes.

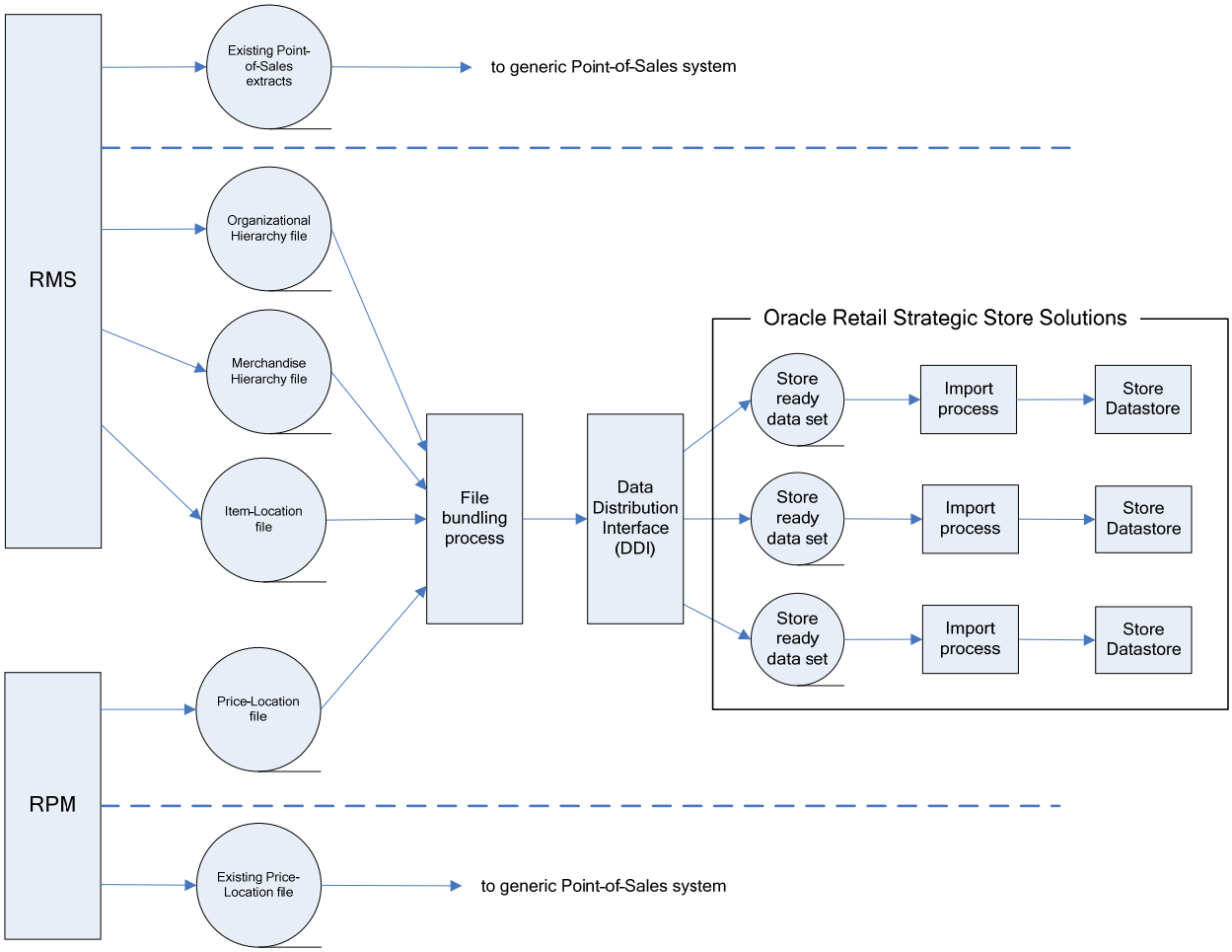
File Details

- Provide one file per store in XML format.
- One file is sent per day
- Each file contains data specific to that store only.
- Each file contains data for items that have changed price since the last file was created for that store.

Integration Dataflow

The diagram below details the overall direction of the dataflow among the various systems. The accompanying explanations of these diagrams are written from a system-to-system perspective, illustrating the movement of data throughout the RPM-related portion of the enterprise.

RPM/Oracle Retail Strategic Store Solutions Communication Flow Diagram



Functional Description of Dataflow

From RPM to ORBO

RMS and RPM pass data to Oracle Retail Back Office (ORBO). RPM passes pricing data. This data is combined with organizational hierarchy, merchandise hierarchy, and item data from RMS. The data is bundled, reorganized by store, and then sent to ORBO.

RPM creates the following data files for ORBO:

File	Data (in the file Stores will consume)	Description	Full Load or Incremental
Price	Promotion data, Clearance data, and Regular price data	Contains promotion data, clearance data, and regular price data.	Incremental

Data Bundling

The data-bundling process within RPM reads the price location data and bundles it to create separate files for each ORPOS store.

Data bundling specific to the RRPM to Oracle Retail Strategic Store Solutions integration is done by jarring the XML files generated by SQL extract scripts. This jarring (bundling) is performed by the batch_orpos_extract.ksh. This extract batch also creates a manifest file that defines the interdependencies of the XML files and is included in the bundle too. Here is the flow:

1. batch_orpos_extract.ksh runs and calls the SQL extract scripts.
2. SQL extract scripts generate the XML files needed by ORPOS.
3. Batch_orpos_extract.ksh checks these files for valid data and creates the manifest file.
4. Batch_orpos_extract.ksh creates the bundle via jar command and clears the temporary files.

Known Issues

Mismatch in promotion functionality

There is a mismatch in promotion functionality between what RPM supports and what Oracle Retail Strategic Store Solutions supports. The promotion types that RPM supports that are not currently supported by Oracle Retail Strategic Store Solutions are listed below. If the user creates one of these promotion types, it will not be sent to Oracle Retail Strategic Store Solutions, because it does not fit in the current model of the XML report.

- BuyGet promotions with Buy type are all excluded.
- Threshold promotions with more than one level are excluded (for example, qty1=20% off, qty 2=30% off, qty 3=40% off). Only threshold promotions with one level are sent to Stores.
- Threshold promotions with a threshold type of Amount are excluded.
- Threshold promotions with a qualification type of Item level are excluded.

Other gaps between RPM and Oracle Retail Strategic Store Solutions

- While multi-unit pricing can be set up in RPM, it is out of scope for ORPOS integration.
- Fixed-price price changes and promotions can be set up with a unit of measure (UOM) other than EA (eaches). However, UOM is not sent to Oracle Retail Strategic Store Solutions on the Pricing extract file.
- RPM clearance price changes are treated the same as regular price changes; Oracle Retail Strategic Store Solutions does not distinguish the clearance price change type.

5

Functional Design

Overview

This chapter provides information concerning the various aspects of RPM's functional areas. Topics include:

- Functional assumptions
- Functional overviews
- Concurrency considerations

Functional Assumptions

- Initial price setting does not respect link codes.
- The conflict review list is displayed when the user approves a primary area of an area differential price change. The only way to determine if the primary area caused the conflict or the secondary area caused the conflict is by the price change ID.
- RPM uses RMS's VDATE to represent today's date rather than the server's system date.
- Reason codes for price changes and clearances should be no longer than 2 characters in length.
- RPM recognizes only sellable items.
- If the retailer includes VAT as part of the retail, VAT regions, VAT items, and VAT codes must be set up in the merchandising system (such as RMS).
- The price change dialog area differential functionality has *not* been modified to perform immediate price adjustments for secondary areas based on changes to the primary area. This is new functionality that has been added to the worksheet. In addition, the area differential competitor logic has not been incorporated in the price change dialog.
- All market basket codes (competitive and margin) must be entered and maintained in a codes table by a database administrator (DBA). There is no associated user interface (UI) to enter and/or maintain the codes.
- Constraint checks are not performed for item/locations automatically added (due to a new item/location relationship) to a price change, clearance, or promotion. In addition, when a location moves from one zone to another zone, or a location is added to a zone, there are no promotion constraint checks performed when the new location is added to the existing zone level events.
- Promotion constraint checks are not performed against price changes requested by SIM.

Functional Overviews

Zone Structures

Zone structures in RPM allow the client to define groupings of locations for pricing purposes and eliminate the need to manage pricing at a location level. At the highest level, these groupings are divided into categories called ”zone groups”. While definition of these zone groups is flexible, primarily it is according to pricing scheme. The three types of zone groups in RPM are regular zone groups, clearance zone groups, and promotion zone groups.

In addition to being defined by pricing, zone groups are defined by the item(s) being priced. The following are examples of zone groups:

- Regular price beverage zone group
- Regular price footwear zone group
- Promotion price beverage zone group

Within zone groups in RPM are groupings of locations (stores and/or warehouses) called “zones”. The function of these zones is to group locations together in a manner that best facilitates company pricing strategies. Zone definition is flexible. For example, the client may choose to create zones based on geographic regions, such as the following:

- US East region
- US West region
- Mexico stores

Similarly, the client may create zones with locations that share similar characteristics, such as the following:

- US urban stores
- US rural stores

Contained within zones are ‘locations’. These locations can be stores or warehouses. There are no restrictions on the number of locations a zone can contain. However, there are two rules that apply to the relationship between locations and zones.

1. A location cannot exist in more than one zone within a zone group. A location can, however, exist in multiple zone groups. For example, a New York City store might exist in the US urban stores zone group as well as the US East region zone group.
2. All locations within the same zone must use the same currency.

> **Note:** When locations are deleted from an existing zone, RPM handles this processing in the same way that it handles location moves processing by deleting all future retail data for that zone/location. See ”Location Moves” in this chapter for more information.

Once zone groups have been created in RPM, users are able to assign them to primary zone group definitions. The primary zone group definition allows the user to specify the zone structure to use when pricing merchandise hierarchies--and how to initially price items in these hierarchies (markup %, markup type). These definitions can be created at the department, class, or subclass level.

Codes

Market Basket Codes

A market basket code is a mechanism for grouping items within a hierarchy level to apply similar pricing rules (margin target or competitiveness). Market basket codes cannot vary across locations in a zone. RPM thus assigns and stores market basket codes against an item/zone. An RPM user can set up the following two market basket codes per item/zone:

- One used in conjunction with the competitive pricing strategy (competitive market basket code).
- One used in conjunction with the margin and maintain margin pricing strategy (margin market basket code).

When the merchandise extract batch process runs, the program identifies the pricing strategy being executed and uses the items extracted, the zone information on the strategy, and the type of strategy to determine what market basket codes to use when proposing retails.

Link Codes

Link codes are used to associate items to each other at a location and price them exactly the same. RPM users set up and maintain item/location link code assignments.

Price Changes, Promotions, Clearances, and Promotion Constraint

Overview

Pricing events in RPM are broken into three primary categories. They are:

- Price changes
- Promotions
- Clearances

Although these pricing activities are unique, they have a common look and feel. This section also addresses promotion constraint functionality, where users are warned if they are creating a price change within a set number of days of the start of an approved promotion.

Price Changes

Price changes are the pricing events in RPM that affect the regular retail price. There are several factors, such as competitor pricing and desired profit margin, that compel retailers to create a manual price change. When a price change is created, the client specifies the following:

- Which item is receiving the price change
- Where the price change is occurring
- How the price of the item is changing
- When the price change will take effect

The highest item level that a price change can be applied to is the parent level, and the highest location level that a price change can be applied to is the zone level. The client has the option of creating exceptions to price changes created at one of these higher levels (such as, mark all men's turtleneck sweaters down 10%, but mark the large size down 5%).

When price changes are approved in RPM, they are published to RMS for ticketing purposes. The night before an approved price change is scheduled to take effect, RMS pricing information is updated with the new regular retail resulting from the price change.

Promotions

Although promotions and price changes have similar characteristics, there are several factors that distinguish these pricing events. The promotion of items is frequently driven by a particular event, such as a holiday or the overstock of an item.

When a promotion is entered in RPM, the client specifies the duration of the promotional price, what kind of promotion will take effect, and to which item(s)/location(s) the promotional price applies. Unlike price changes, where the highest item level is parent, the highest level a promotion can be set at is the department level (for example, Men's clothing). Promotions can be set up to apply to the regular retail price, the clearance retail price, or both; when the promotion ends, the price reverts back to the retail price that existed prior to the promotion.

Exceptions to promotions also can be created at one of these higher levels (such as, mark all men's turtleneck sweaters down 10%, except mark down the large size 5%). Specific item/locations also can be excluded from a promotion.

Examples of promotion types in RPM are:

- Complex promotion: Buy 3 shirts, get 1 free
- Simple promotion: 25% off the retail price of an item
- Threshold promotion: Spend $100, get $10 off.

Clearances

Clearances in RPM are markdowns or a series of markdowns designed to increase demand and therefore move inventory out of a store. Each subsequent clearances always results in a lower price for the item. When a clearance is created, the client specifies the item(s) and locations where the clearance is in effect and the discount or set price for the markdown.

Clearances can be applied at the following item levels: parent, parent/differentiator, and transaction level. Clearances can be applied at the location or price zone level.

When a clearance price is created, the client specifies a reset date on which the clearance price reverts back to regular retail price. Reset dates are optional.

> **Note:** Promotion and clearance events are not communicated to RMS for the purpose of ticketing.

Promotion Constraint

Users are warned if they are creating a price change within a set number of days of the start of an approved promotion or vice versa. This warning does *not* stop the user from creating the price change or promotion. The number of days is determined by a promotion constraint variable that is stored at the subclass/location level.

When the user runs conflict checking on a price change record, promotion record, or worksheet status record, promotion constraint checks are run. If a promotion constraint is violated, the user has the option to either ignore the constraint, or have the system help suggest a date that does not violate the constraint (price change and worksheet dialogs only). The user is also able to optionally select to ignore promotion constraints on individual price change and worksheet detail records so promotion constraint checks are never performed when conflict checking is run.

Pricing Strategies

The pricing strategies front end allows the client to define how item retails are proposed when pricing worksheets are generated. The strategies can be defined at department, class or subclass. The strategies are grouped into two categories, regular and clearance. An item/location can be on one "maintain margin" strategy and on one other strategy.

When setting up pricing strategies, the first task is to specify what type of pricing strategy is to be applied--and at what merchandise hierarchy/location hierarchy combination. The pricing strategy types are described in this functional overview and include the following:

- Area differentials and competitor strategies
- Clearance
- Competitive
- Margin
- Maintain margin

When determining what merchandise level a pricing strategy will be applied to, the lowest definable level (from aggregation) is taken into consideration.

Other contributing factors when establishing pricing strategies include attaching price guides to the strategy, specifying a calendar to run against the pricing strategy, and attaching warehouses (if they are not recognized as pricing locations) to a strategy for the purpose of inventory visibility.

Area Differentials and Competitor Strategies

Area differential pricing allows a buyer to perform the following:

- Set prices for items against a primary zone.
- Price other zones off of the primary zone using a defined differential.

This functionality allows the user to focus on establishing a primary retail, which drives system-generated prices for other secondary areas.

Secondary area retails can change based on primary area changes, regardless of the status of the primary areas. All secondary area proposed price changes are created in worksheet status to allow for immediate updates. To allow the user to work through multiple zone worksheets at the same time (primary zone included), the system provides a dynamic update of the worksheet changes from the primary area to the secondary area. Following the same logic in a ”what if” scenario, price changes are not actually created in the worksheet dialog until the user approves a worksheet. This approach is available only in the Worksheet status and Worksheet detail dialogs. A system option signifies whether this process is used. This functionality does not exist in the price change dialog.

Layered Competitive Pricing Comparisons in the Worksheet

In the worksheet, competitive pricing comparisons are ”layered”on top of the area differential pricing rules for secondary areas. Note that layering is not carried out throughout the RPM application. The area differentials front end allows the user to set up and maintain competitive pricing strategies that are specifically associated to the differential.

The area differential price acts as a guide. The proposed retail based on the defined area differential is compared to the proposed retail based on the competitor rules defined. The lower of the two retails is the retail that RPM proposes. After the resulting retails are compared and the lower one is chosen, the pricing guidelines are applied.

Scenarios

This section illustrates the retails that are derived under various scenarios.

Setup Data

The following setup data applies to all the scenarios.

	Zone Group ID	Zone	Rule	%
Primary	1000	1000		
Secondary	1000	2000	Percent Higher	5
Secondary	1000	3000	Percent Lower	5

Scenario 1

Note that the proposed retails are displayed based on the percent higher or lower for the secondary zones.

Primary	Margin Strategy:	10%
	Item A Cost:	4.55
Secondary	Competitor Strategy:	None
	Pricing Guide:	None

	Item	Zone	Basis Retail	Proposed Retail
Primary	A	1000	4.7	5
Secondary	A	2000	4.8	5.25
Secondary	A	3000	4.9	4.75

Scenario 2

Note that because the basis retail for Item A in zone 1000 was already $5.00, no proposed retail is displayed. The secondary area proposed retail was calculated using the current retail of the primary area.

Primary	Margin Strategy:	10%
	Item A Cost:	4.55
Secondary	Competitor Strategy:	None
	Pricing Guide:	None

	Item	Zone	Basis Retail	Proposed Retail
Primary	A	1000	5	
Secondary	A	2000	4.8	5.25
Secondary	A	3000	4.9	4.75

Scenario 3

Note that the proposed retails all end in 6 due to the pricing rule in place.

Primary	Margin Strategy:	10%
	Item A Cost:	4.55
Secondary	Competitor Strategy:	None
	Pricing Guide:	Ends in 6

	Item	Zone	Basis Retail	Proposed Retail
Primary	A	1000	4.7	5.06
Secondary	A	2000	4.8	5.26
Secondary	A	3000	4.9	4.76

Scenario 4

Note the proposed retails for the secondary areas are equal to the competitor price due to the competitor strategy in place.

Primary	Margin Strategy:	10%
	Item A Cost:	4.55
Secondary	Competitor Strategy:	Match
	Pricing Guide:	None
	Competitor Retail:	4.7

	Item	Zone	Basis Retail	Proposed Retail
Primary	A	1000	4.7	5
Secondary	A	2000	4.8	4.7
Secondary	A	3000	4.9	4.7

Scenario 5

Note the proposed retail for zone 3000 did not change to $4.85 even though the competitive strategy is match and the competitor retail is $4.85. The competitor strategy should never cause a price to increase.

Primary	Margin Strategy:	10%
	Item A Cost:	4.55
Secondary	Competitor Strategy:	Match
	Pricing Guide:	None
	Competitor Retail:	4.85

	Item	Zone	Basis Retail	Proposed Retail
Primary	A	1000	4.7	5
Secondary	A	2000	4.8	4.85
Secondary	A	3000	4.9	4.75

Scenario 6

Note the pricing rule was applied after the competitive strategy causing the competitor price to be matched to the $4.85 and then adjusted to $4.86 to account for the pricing guide.

Primary	Margin Strategy:	10%
	Item A Cost:	4.55
Secondary	Competitor Strategy:	Match
	Pricing Guide:	Ends in 6
	Competitor Retail:	4.85

	Item	Zone	Basis Retail	Proposed Retail
Primary	A	1000	4.7	5
Secondary	A	2000	4.8	4.86
Secondary	A	3000	4.9	4.76

Clearance Strategy

The clearance strategy allows the client to define the markdowns that should be proposed for items in the specified merchandise hierarchy/location hierarchy level. Each markdown specified in the strategy has an associated markdown percent. The strategy also defines to what retail the markdown percent should be applied (regular retail or clearance retail). This section illustrates the retails that are derived under various scenarios.

Scenario 1

Markdown Number	Markdown Percent
1	20
2	30
3	50
4	75

Item/Zone is currently not on clearance.
Regular Retail: $35.00
Clearance Retail: n/a
Proposed Markdown/Retail for Item/Zone: Markdown 1/$28.00

Item/Zone is currently on its first markdown.
Regular Retail: $35.00
Clearance Retail: $28.00
Proposed Markdown/Retail for Item/Zone: Markdown 2/$24.50

Item/Zone is currently on its second markdown.
Regular Retail: $35.00
Clearance Retail: $24.50
Proposed Markdown/Retail for Item/Zone: Markdown 3/$17.50

Item/Zone is currently on its third markdown.
Regular Retail: $35.00
Clearance Retail: $17.50
Proposed Markdown/Retail for Item/Zone: Markdown 4/$8.75

Scenario 2

Apply To Type: Clearance Retail

Markdown Number	Markdown Percent
1	25
2	10
3	10
4	10

Item/Zone is currently not on clearance.
Regular Retail: $35.00
Clearance Retail: n/a
Proposed Markdown/Retail for Item/Zone: Markdown 1/$26.25

Item/Zone is currently on its first markdown.
Regular Retail: $35.00
Clearance Retail: $26.25
Proposed Markdown/Retail for Item/Zone: Markdown 2/$23.63

Item/Zone is currently on its second markdown.
Regular Retail: $35.00
Clearance Retail: $23.63
Proposed Markdown/Retail for Item/Zone: Markdown 3/$21.27

Item/Zone is currently on its third markdown.
Regular Retail: $35.00
Clearance Retail: $21.27
Proposed Markdown/Retail for Item/Zone: Markdown 4/$19.14

Competitive Strategy

This strategy allows the client to define which competitor's retails to reference and how to make comparisons to those retails when proposing new retails.

In other words, the competitive strategy allows you to define the following:

- A primary competitor retail that should be referenced when proposing retails for items in the specified merchandise hierarchy/location hierarchy level.
- How to propose new retails based on that primary competitor's retail (for example, price above a certain percent, price below a certain percent, or match the competitor's retail).

This section illustrates the retails that are derived under various scenarios.

Scenario 1

Regular Retail for Item/Zone: $26.00
Primary Competitor Retail for Item: $25.00
Strategy: Price Above – 10%
Acceptable Range: 8% - 12% Above
Proposed Retail: 27.50

Scenario 2

Regular Retail for Item/Zone: $27.50
Primary Competitor Retail for Item: $25.00
Strategy: Price Above – 10%
Acceptable Range: 8% - 12% Above
Proposed Retail: No proposal

Scenario 3

Regular Retail for Item/Zone: $22.50
Primary Competitor Retail for Item: $25
Strategy: Price Below – 10%
Acceptable Range: 8% - 12% Below
Proposed Retail: No proposal

Scenario 4

Regular Retail for Item/Zone: $21.00
Primary Competitor Retail for Item: $25
Strategy: Price Below – 10%
Acceptable Range: 8% - 12% Above
Proposed Retail: 22.50

Scenario 5

Regular Retail for Item/Zone: $28.00
Primary Competitor Retail for Item: $25
Strategy: Match
Acceptable Range: n/a
Proposed Retail: $25.00

Margin Strategy

This strategy allows the client to define the target amount of markup above cost (margin target). The system uses this value to propose new retails for the items in the specified merchandise hierarchy/location hierarchy level. This section illustrates retails derived under various scenarios

Scenario 1

Regular Retail for Item/Zone: $25.00
Cost of the Item: $18.00
Margin Target: 25%
Acceptable Range: 23% - 27%
Markup Type: Retail
Proposed Retail: $24.00

Scenario 2

Regular Retail for Item/Zone: $23.50
Cost of the Item: $18.00
Margin Target: 25%
Acceptable Range: 23% - 27%
Markup Type: Retail
Proposed Retail: No proposal

Maintain Margin Strategy and Auto Approve

The maintain margin strategy allows a retailer to receive proposed retail changes based upon impending cost changes. Proposed retail changes depend on either the retail margin or cost margin. The formulas and calculations for both methods are illustrated later in this overview.

The maintain margin strategy retrieves all cost changes related to a specified zone/merchandise hierarchy on a look-forward basis and generates proposed retail changes. In the unlikely event there are multiple cost changes in the forward-looking review period, the system bases the proposed retail on the last cost change to occur during that forward-looking review period. The retail changes proposed can be based on the current margin percent between the item retail and the cost, or the market basket code margin associated with the item. The user also can specify how to apply the increase/decrease in retail in one of two ways:

- As a margin percent (current or market basket) applied to the new cost.
- As the monetary amount change to the cost applied to the retail (for example, a 5 cent increase in cost results in a 5 cent increase in retail).

Reference competitors can be attached to the maintain margin strategy. Note, however, that these competitors do not drive price proposals. They are visible via the worksheet once a price change proposal has been created.

Merch Extract Calculations

> **Note:** For all calculations below, if price guides are assigned to the strategy, they are applied after the calculations above have been completed.

Market basket margin

Cost method:

Proposed retail = ((New Cost * Margin Target%) + New Cost) + VAT rate if applicable.

Retail method:

Proposed retail = (New Cost / (1- Margin Target%)) + VAT rate if applicable

Current margin

Current margin % =

Cost method: (Retail on Review period start date – Cost on review period start date) / Cost on review period start date

Retail method: (Retail on Review period start date – Cost on review period start date) / Retail on review period start date

Using the current margin % calculated above, the retail can be proposed.

Cost Method

Proposed retail = ((New Cost * Current Margin %) + New Cost) + VAT rate if applicable.

Retail Method:

Proposed retail = (New Cost / (1- Current Margin%)) + VAT rate if applicable

Change by cost change amount

Proposed retail = (New cost – Cost on the day before the New Cost date) + Retail on day before the New Cost Date. This retail value will include VAT if applicable.

Examples:

Market Basket %	40
Current Margin % (cost)	50%
Current Margin % (retail)	33%
Current Retail	0.75
Current Cost	0.5

Cost method

Method		Application Option		Future		Calculation
Market Basket %	Current Margin %	Margin %	Change by Cost Change Amount	Future Cost	Future Retail	
X		X		0.55	0.77	(0.55*40%) + 0.55
	X	X		0.55	0.61	(0.55*50%) + 0.55
	X		X	0.55	0.8	(0.55-0.5) + 0.75

Retail method

Method		Application Option		Future		Calculation
Market Basket %	Current Margin %	Margin %	Change by Cost Change Amount	Future Cost	Future Retail	
X		X		0.55	0.92	(0.55/(1-.4))
	X	X		0.55	0.61	(0.55/(1-..33))
	X		X	0.55	0.80	(0.55-0.5) + 0.75

Price Inquiry

Price inquiry allows retailers to retrieve the price of an item at an exact point in time. This price may be the current price of a particular item or the future price. Users can search for prices based on the following search criteria:

- Merchandise hierarchy
- Item
- Zone group
- Zone
- Location
- Location (warehouse or store)
- Date

After the search criteria has been specified, a list of item/location combinations are displayed with their corresponding dates--and prices on those dates. Included are the item/location regular, clearance, and promotion prices. Items can be retrieved at the parent, parent/differential, or transaction level, and valid locations are price zone or location. After retrieving the desired pricing information, the user has can export the information outside of the system. For more information regarding the Price inquiry dialog, see the Oracle Retail Price Management User Guide.

Worksheet

The RPM worksheet functionality allows for the maintenance of automatically-generated price change and clearance proposals resulting from the RPM merchandise extract batch program. These proposed price changes/clearances are the product of existing strategies, calendars and item/location information. The merchandise extract program outputs them to worksheet at the transaction item level for zone. Worksheet groups these values together. While not all items have a proposed price change, each item in the pricing strategy is represented.

The worksheet dialog has two main screens: the worksheet status screen and the worksheet detail screen. In the worksheet status screen, each row of the table represents an individual worksheet. Depending on the status of the worksheet--and the records within the worksheet--the following actions may be performed:

- Submit
- Approve
- Reject
- Reset
- Delete

> **Note:** Actions affect only those records that can be affected. For example, if a record is already in "approved" status, the Submit command will not apply to that record.

The Worksheet detail form displays information about the records contained in a given worksheet. Among the information displayed is the proposed price change generated by the merchandise extract program and other information that can assist in making determinations on whether to accept, reject, or modify proposed pricing. After these determinations have been made, the worksheet can be submitted for approval, rejection, and so on.

A worksheet does not exist until the merchandise extract program has run, when the status of the worksheet becomes New. However, the exception to this process is for those items/zones involved in an area differential pricing strategy. The locations that are part of a secondary area have a worksheet and corresponding rows in either pending or new status based on the Dynamic Area Differential system option. If the system option is unchecked, the secondary areas contain no detailed information for individual rows until price changes for the primary area are approved. If the system option is checked, the secondary areas contain detailed retail information derived from the retails proposed for the primary area.

Merchandise Extract

The merchandise extract batch process creates worksheets based on calendars and pricing strategies.

The three-step process does the following:

1. Identifies the work to be processed.
2. Extracts RMS data into RPM.
3. Uses extracted data to propose retails (based on strategies) and build the worksheets.

The merchandise extract batch program initially finds calendars (with review periods that start tomorrow) and the pricing strategies that use those calendars. This processing determines which item/locations are pulled into the worksheet. There are attributes associated with calendar review periods, and these help to determine whether candidate rules or exceptions are run for that particular review period.

Candidate rules are run against the items/locations being extracted from the merchandise system to determine if they should be flagged for review. They are defined at the corporate level and can contain variables at the department level. Candidate rules can be inclusive or exclusive. If they are inclusive, and the candidate rule is met, the item/location is flagged in the worksheet. When exclusive candidate rules are met, the item/location is excluded from the review when the merchandise extract program builds the worksheet. Candidate rules can also be active or inactive, allowing the user to suspend rules that are only needed at certain times of the year. Candidate rules are only run against the worksheet the first time the worksheet is created.

Exceptions: Each review period has an indicator stating whether or not to run exceptions. If the indicator is set to 'Y'es, the merchandise extract should tag those Item/Location records that are pulled into the worksheet with an exception flag if any of the following occur during a review period where exceptions are processed: competitor regular retail price changes, cost changes, and new item/location relationships.

For every item/location pulled into the worksheet, RPM attempts to propose a new retail based on the strategy attached to that item/location. When the worksheet is first created, the details of the strategy are saved. Updates to the strategy do not affect any worksheets that are currently being reviewed. The updates are only reflected in worksheets generated after the updates to the strategy are made. Until the worksheet has been locked, new retails should continue to be proposed using the strategy details every night the batch program is run.

Below is a list of reasons why an item/location does not appear in the worksheet:

- The item does not have a record on the future cost table for the location on a margin strategy.
- There are varying link codes across the item/locations.
- The strategy is set up at a zone level, and the unit of measure for the item varies across the locations in the zone.
- The merchandise extract program is running a margin strategy or competitive strategy against a zone, and all of the locations within the zone do not share the same market basket code.
- The merchandise extract program in running against a strategy set up at the zone level (where the zone is not the primary zone for the item), and all of the locations within that zone do not share the same BASIS selling unit of measure.
- An exclusion candidate rule has been met.
- The item is not ranged in the location, or the strategy is at the zone level and the item is not ranged to any location in that zone.
- The items is on the exclude list of an area differential strategy.

See "Chapter 6 – Java and RETL Batch Processes" for additional information about this batch process.

Calendar

Calendars are set up in RPM for the primary purpose of attaching them to pricing strategies. When a calendar is created in RPM, a start date is selected initially. This date can be no earlier than tomorrow. In addition, for the calendar to be valid, an end date that is later than the start date must be specified.

Once the time frame of the calendar has been established, review periods for the calendar can be specified (in numbers of days) The number of days between those review periods also can be specified. Collectively, these periods span the date range of the new calendar. When establishing the review period duration, the review periods and the time between them must exactly reach the specified end date. If these actions are not performed properly, RPM suggests an end date that makes the calendar valid. The following is an example of a valid calendar:

Start Date: 01/01/04
End Date: 01/20/04
Review Period Duration: 3 Days
Days Between Review Periods: 3 Days

Start Date	End Date
01/01/04	01/03/04
01/07/04	01/09/04
01/13/04	01/15/04
01/19/04	01/21/04

Aggregation Level

Aggregation level functionality is used in RPM to define parameters that vary at the department level. Within this functional area, users select a department and specify the lowest definable level at which the pricing strategies can be defined. The merchandise hierarchy levels at which a pricing strategy can be defined are department, class, and subclass.

When the merchandise extract runs to generate worksheets, the Worksheet Level setting is used to determine the level at which the worksheets should be generated. Merchandise hierarchy levels with varying strategies can be aggregated into the same worksheet based on this aggregation level setting. For example, the strategies for a worksheet may be defined at the class level, but if the worksheet level for the department that class is in is set to department then a single worksheet status row exists per zone with all the classes rolled up to the department.

The sales settings on the aggregation level screen determine the sales types that are pulled during the extract process and represented in the worksheet as historical sales. The inventory settings determine how warehouse inventory is utilized and which inventory the sell through calculations use.

Location Moves

Location moves in RPM allow users to select a location that exists in a zone and schedule it to move to a different zone within a zone group on that scheduled date. The process of moving a location uses conflict checking to fix potential pricing problems. After conflict checking is complete, the process also allows the location to save most valid pricing events through the move and to smoothly transition out of their old zone pricing strategies into the new zone pricing strategies. Conflict checking identifies the following:

- Strategies with the old zone/new zone attached
- Other scheduled location moves for that location
- Promotions at the old zone/new zone that span the location move

> **Note:** If any of the above is found during conflict checking, the location move fails.

When a location move is successfully scheduled in RPM, all future retail data for the old zone/ location is removed. Location level pricing events remain intact, but exclusions are created if the new zone pricing events create conflicts such as a negative retail.

A retail for a location does not change when it moves into the new zone, but the new zone pending pricing events will be applied to the location after the location move date.

Concurrency Considerations

This section contains currency considerations and solutions within the RPM system. If multiple users are using the same data, RPM has concurrency solutions to prevent the persistence of invalid or inaccurate data in the RPM database.

Pessimistic Data Locking

Pessimistic locking prevents data integrity issues that are missed by business rules/validation due to overlapping transactions.

For example, two users working on the same set of data kick off the approval process for a price event. If the second process is started after the first process has completed, the application business rules will handle the concurrency issues.

Although this scenario only arises in a very specific case within a specific period, ramifications of the resulting overwrites can be severe due to the loss of data integrity (especially with respect to retails going below zero, events at locations that have been moved or are scheduled to be moved, incorrect basis value retails, and so on).

With pessimistic locking, the first user in locks the data until he or she is finished with that transaction. If a second user tries to lock the same data, he or she is notified that the data is currently locked by another user. Because a user can only update data that he or she has locked, data integrity is guaranteed.

Pessimistic Workflow Locking

With pessimistic workflow locking prevents editing within a workflow that is currently in use by another user.

Scenario:

Two pricing managers have security access to the same department for price change decisions. User one selects a worksheet in the worksheet status screen for Dept 100 in Zone 100 and opens the detail screen. User two enters the worksheet status screen and attempts to review the same worksheet. The system does not allow user two to open that worksheet and issues a message that the worksheet is in use by another user.

Last User Wins

Data submitted by the second user overwrites data submitted by the first user. With Last User Wins, there is no warning or message to notify users that they have overwritten data modified by previous users.

If subsequent changes are incompatible with previous changes, business validation/rules protect data integrity. In this case, the second user will receive the appropriate business exception message.

Scenario*:*

Two users have been told to update a pricing strategy. User one enters the strategy and changes the value. User two enters the strategy. Since user one has not saved the change yet, user two still sees the original value and makes the change. User one then saves the change and leaves the dialog. User two then saves a change. Since there is no validation that has been broken, the second user change also is saved, resulting in no difference from the first user. If the second user changed the value and validation failed, that user is prompted with an error to fix the problem, just as though the second user created the validation error.

Optimistic Data Locking

The second user receives an error message if an attempt is made to overwrite data modified by the first user. The message indicates the user has been working with stale data and should re-load and re-process the changes. With Optimistic Data Locking, the first user wins which is the opposite of the Last User Wins approach.

Concurrency Solution/Functional Area Matrix

	Pessimistic Data Locking	Pessimistic Workflow Locking	Last User *Wins*	Optimistic Data Locking
Clearance Price Changes	X		X	
Price Changes	X		X	
Promotions	X		X	
Future Retail/Conflict Checking	X		X	
Location Moves	X		X	
Worksheet		X		
Aggregation Level			X	
Area Differentials			X	
Calendar			X	
Candidate Rules			X	
Foundation			X	
Initial Price Settings			X	
Pricing Attributes			X	
Pricing Guides			X	
Pricing Strategies			X	
Promotional Funding			X	
Security			X	
System Options			X	
Zone Structure			X	
Link Codes			X	
Merch Extract			X	
Zone Future Retail			X	

6

Java and RETL Batch Processes

This chapter is divided into two sections. The first section reflects Java-based batch processing within RPM. The second section concerns RETL extract batch processing (for a data warehouse such as RDW, for example)

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). Oracle Retail 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 does 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
clearancePriceChangePublishBatch.sh	ClearancePriceChangePublishBatch
injectorPriceEventBatch.sh	injectorPriceEventBatch
itemLocDeleteBatch.sh	ItemLocDeleteBatch
itemReclassBatch.sh	itemReclassBatch
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.
locationMoveBatch.sh	LocationMoveBatch
merchExtractKickOffBatch.sh	MerchExtractKickOffBatch
newItemLocationBatch.sh	NewItemLocationBatch
PriceChangeAreaDifferentialBatch.sh	PriceChangeAreaDifferentialBatch
priceChangeAutoApproveResultsPurgeBatch.sh	PriceChangeAutoApproveResultsPurgeBatch
priceChangePurgeBatch.sh	PriceChangePurgeBatch

Script	Batch program executed
priceChangePurgeWorkspaceBatch.sh	PriceChangePurgeWorkspaceBatch
priceEventExecutionBatch.sh	PriceEventExecutionBatch
priceStrategyCalendarBatch.sh	PriceStrategyCalendarBatch
promotionPriceChangePublishBatch.sh	PromotionPriceChangePublishBatch
promotionPurgeBatchbatch.sh	PromotionPurgeBatchbatch
purgeBulkConflictCheckArtifacts.sh	purgeBulkConflictCheckArtifacts
purgeExpiredExecutedOrApprovedClearancesBatch.sh	PurgeExpiredExecutedOrApprovedClearancesBatch
purgeLocationMovesBatch.sh	PurgeLocationMovesBatch
purgeUnusedAndAbandonedClearancesBatch.sh	PurgeUnusedAndAbandonedClearancesBatch
regularPriceChangePublishBatch.sh	RegularPriceChangePublishBatch
RPMtoORPOSPublishBatch	RPMtoORPOSPublishBatch
RPMtoORPOSPublishExport	RPMtoORPOSPublishExport
statusPageCommandLineApplication.sh	statusPageCommandLineApplication
taskPurgeBatch.sh	TaskPurgeBatch
worksheetAutoApproveBatch.sh	WorksheetAutoApproveBatch
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 batch processes.

Batch processes	Details
ClearancePriceChangePublishBatch	This batch process formats and stages output of clearance price change price events.
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.

Batch processes	Details
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.
NewItemLocationBatch	The NewItemLocationBatch process is utilized when you are operating RPM without the RIB. This batch process 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.
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.
PromotionPriceChangePublishBatch	This batch process formats and stages output of promotion price change price events.
PromotionPurgeBatchbatch	This batch process deletes old and rejected promotions.
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
PurgePayloadsBatch	This batch program purges entries related to price events from the RPM_*PAYLOAD tables.
PurgeUnusedAndAbandonedClearancesBatch	This batch process deletes unused and rejected clearances.

Batch processes	Details
RefreshPosDataBatch	This batch generates all pricing information (current and future) for a store. The information is then sent to POS for the purpose of refreshing POS systems data.
RegularPriceChangePublishBatch	This batch process formats and stages output of regular price change price events.
RPMtoORPOSPublishBatch	The RPMtoORPOSPublishBatch program formats and stages output of different price events like PriceChange, Clearance and Promotions.
RPMtoORPOSPublishExport	The RPMtoORPOSPublishExport program calls a SQL script "RPMtoORPOSSpoolMsg.SQL". This script spools the data collected from different publish tables of different price events (Price Change, Clearance and Promotion).
statusPageCommandLineApplication	The status page batch program (statusPageCommandLineApplication.sh) performs some data checks, to verify that some of the assumptions that the application makes about the data are not violated.
TaskPurgeBatch	The TaskPurgeBatch purges the entries from RPM_*TASK tables based on the entered purge days and the status indicator.
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 process deletes past zone/item price change actions.

Batch Process Scheduling

Before setting up an RPM process schedule, familiarize yourself with the 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 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".

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.

1. 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.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

7. 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.

8. 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).

9. 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.

10. 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

11. 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.

12. 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:

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

   If the indicator is set to Yes (checked), the retail price can be 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.

2. A regular price chance 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.

3. 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 loops 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 than 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 the 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 a problem during the publishing, the whole collection will be rolled backed.
- Failures in the following areas (which are executed in the same transaction of the conflict checking process) result in the roll back of the entire collection in a thread (see explanation on Threading mechanism below):
  - Purge of old Future Retails records
  - Population of the payload tables
  - Zone Future Retail processing (for primary zone price changes).

  An unexpected error during the transaction (e.g. Table Locking or other Database error) also results in the roll back of the entire collection in a thread.

The following example illustrates how the conflict checking engine processes the collection of price events:

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 ("sequence") these price changes into several collections based on the item/location combination so that there are no item/locations overlapping in the sequence. In the example above, 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 not 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 make this conflict checking process run 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, see the row in the RPM_BATCH_CONTROL table where the value in PROGRAM_NAME column is com.retek.rpm.app.bulkcc.service.BulkConflictCheckAppService.

The value of the THREAD_LUW_COUNT column decides the maximum number of item/locations in the thread. The client 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 current default value 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, the following additional rule that prevents that: Item/locations of one price change cannot be processed across multiple threads.

ClearancePriceChangePublishBatch Batch Design

Overview

The ClearancePriceChangePublishBatch program formats and stages output of clearance price change price events.

The corresponding clearancePriceChangePublishExport shell script produces a pipe ('|') delimited flat-file export based on the output of the ClearancePriceChangePublishBatch.

Usage

The following command runs the ClearancePriceChangePublishBatch job:

```
ClearancePriceChangePublishBatch userid password
```

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

The following command runs the clearancePriceChangePublishExport job:

```
clearancePriceChangePublishExport.sh database-connect-string path
```

Where the first argument is the database connect string (user/pwd@database) and the second argument is the path where the file should be written. The path is optional. If it is not supplied, the path ../output will be used.

Detail

The batch looks for price events in the RPM_PRICE_EVENT_PAYLOAD table with a RIB_FAMILY of "CLRPRCCHG" and distributes those events to multiple threads based on the settings in the RPM_BATCH_CONTROL table. Each thread reads in its set of clearance price change events from tables RPM_PRICE_EVENT_PAYLOAD and RPM_CLEARANCE_PAYLOAD and generates output in RPM_PRICE_PUBLISH_DATA.

PurgePayloadsBatch should be run to delete records in the payload tables.

A single flat-file (CLRPC_<timestamp>.pub) is created, where <timestamp> is the current system time stamp.

Output File

FHEAD – REQUIRED: File identification, one line per file.

FDETL – OPTIONAL: Price Change Event (Create or Modify)

FDELE – OPTIONAL: Price Change Event (Delete)

FTAIL – REQUIRED: End of file marker, one line per file.

Output File Layout

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record Descriptor	Char(5)	FHEAD	File head marker
	Line id	Number(10)	1	Unique line id
	File Type	Char(5)	CLRPC	Clearance Price Changes
	Export timestamp	Timestamp		System clock timestamp (YYYYMMDDHHMISS)
	Format Version	Char(5)	1.0	File Format Version
FDETL	Record Descriptor	Char(5)	FDETL	File Detail Marker (1 per clearance create or modify)
	Line id	Number(10)		Unique line id
	Event Type	Char(3)		"CRE" = Create, "MOD" = Modify
	Id	Number(15)		Clearance identifier
	Item	Char(25)		Item identifier
	Location	Number(10)		Location identifier
	Location Type	Char(1)		'S' = Store, 'W' = Warehouse
	Effective Date	Date		Clearance Effective Date (DD-MMM-YY)
	Selling Retail	Number(20,4)		Selling retail with price change applied
	Selling Retail UOM	Char(4)		Selling retail unit of measure
	Selling Retail Currency	Char(3)		Selling retail currency
	Reset Clearance Id	Number(15)		Id of clearance reset
FDELE	Record Descriptor	Char(5)	FDELE	File Detail Delete Marker (1 per clearance delete)
	Line id	Number(10)		Unique line id
	Id	Number(15)		Clearance identifier
	Item	Char(25)		Item identifier
	Location	Number(10)		Location identifier
	Location Type	Char(1)		'S' = Store, 'W' = Warehouse
FTAIL	Record Descriptor	Char(5)	FTAIL	File tail marker
	Line id	Number(10)		Unique line id
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

Assumptions and scheduling notes

ClearancePriceChangePublishBatch should be run after WorksheetAutoApproveBatch.

ClearancePriceChangePublishExport should be executed as follows:

- After every successful run of ClearancePriceChangePublishBatch.
- Before PurgePayloadsBatch.

Primary tables involved

- RPM_PRICE_EVENT_PAYLOAD
- RPM_CLEARANCE_PAYLOAD

Threading

ClearancePriceChangePublishBatch is threaded. The LUW is a single clearance price change event.

Configuration

The following property must be set in rpm.properties on the application server when using this batch program to publish clearance price events:

```
delete_staged_rib_payloads=false
```

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.

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

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.

Although username and password are optional arguments, they have a predefined position and order in the list of arguments. Username and password should be first in

the list of arguments; all other optional arguments may follow in any order. Except for username and password, the argument identifer should precede each optional argument (for example, event_type=PC).

The following are examples of how applying specific combinations of arguments affects the output of the InjectPriceEventBatch job:

1. Where all arguments are optional

   Arguments: `InjectorPriceEventBatch`

   Result: The batch processes price changes from the staging table in New status, checking the approval process for completion every 10 seconds.

2. Where only username and password are defined by the user

   Arguments: `InjectorPriceEventBatch alain.frecon retek`

   Results: The batch processes price changes from the staging table in New status, checking the approval process for completion every 10 seconds.

3. Where username and password are not defined by the user

   Arguments: `InjectorPriceBatch event_type=CL status=W polling_interval=300`

   Results: The batch processes clearances from the staging table in Worksheet status, checking the approval process for completion every 5 minutes (300 seconds).

4. Where all arguments are defined by the user

   Arguments: `InjectorPriceEventBatch alain.frecon retek event_type=CL status=W polling_interval=300`

   Results: The batch processes clearances from the staging table in Worksheet status, checking the approval process for completion every 5 minutes (300 seconds).

To set up InjectorPriceEventBatch for optional username and password, the following lines should be "commented out" in the launchRpmBatch.sh file:

```
if [ -z "$BATCH_USERNAME" ] ; then
    echo "You must specify a username."
    exit
fi

if [ -z "$BATCH_PASSWORD" ] ; then
    echo "You must specify a password."
    exit
fi
```

Detail

InjectorPriceEventBatch 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 that it must adhere to the predefined importing data format. The contract on the incoming data is defined by the structure of the staging tables on which the batch depends. 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.

Importing Staged Price Changes

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

- No price change exceptions are allowed.
- No parent exceptions are allowed.
- No vendor funding is allowed.
- No deals are allowed.

Other than the field carrying data payload, the table holds fields that facilitate data processing.

- *Auto approve indicator* defines whether the processing batch should attempt to approve the price change after the data has been successfully imported.
- *Status* defines the current state of the data in the staging table, as a result of processing. This status should not be confused with the status of the price event in RPM, even though there are some correlations. Status codes are as follows:
  - N (New): New status indicates that the data hasn't been processed yet
  - W (Worksheet): Worksheet status indicates that the data has been processed and has been imported successfully into RPM. It also indicates that at the time of import, the approval wasn't required
  - A (Approved): Approved status indicates that the data has been imported and approved without any conflicts.
  - E (Error): Error status indicates that the data has been processed, but import has failed due to invalid data. The data administrator can correct the data and rerun the job.
  - F (Failed). Failed status indicates that the data has been imported successfully, but some conflicts have been encountered. The data administrator can correct the data (for example, change the effective date of the event to eliminate the conflict) and rerun the job.

    Note: Initial data should be created in New status. The rest of the statuses are the result of data lifecycle. Records in any status (other than Approved) are eligible for processing by the batch.

- *Error message* holds the message for the first error encountered while importing data. The field actually holds an error message key rather that the actual error message. When conflicts are encountered during approval attempt, the field holds the"CONFLICT_EXISTS" key.
- *Process id* uniquely identifies the batch run. The field is populated for records that are not in New status.
- *Price change display id* is populated only upon 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 structure is similar to that of `RPM_CLEARANCE` but with these limitations:

- No clearance exceptions are allowed.
- No reset records are allowed.
- No vendor funding is allowed.
- No deals are allowed.

Other than the field carrying data payload, the table holds fields that facilitate processing.

- *Auto approve indicator* defines whether the processing batch should attempt to approve the clearance after the data has been successfully imported.
- *Status* defines the current state of the data in the staging table, as a result of processing. This status should not be confused with the status of the price event in RPM, even though there are some correlations. Status codes are as follows:
  - N (New): New status indicates that the data hasn't been processed yet
  - W (Worksheet): Worksheet status indicates that the data has been processed and has been imported successfully into RPM. It also indicates that at the time of import, the approval wasn't required
  - A (Approved): Approved status indicates that the data has been imported and approved without any conflicts.
  - E (Error): Error status indicates that the data has been processed, but import has failed due to invalid data. The data administrator can correct the data and rerun the job.
  - F (Failed). Failed status indicates that the data has been imported successfully, but some conflicts have been encountered. The data administrator can correct the data (for example, change the effective date of the event to eliminate the conflict) and rerun the job.

    Note: Initial data should be created in New status. The rest of the statuses are the result of data lifecycle. Records in any status (other than Approved) are eligible for processing by the batch.

- *Error message* holds the message for the first error encountered while importing data. The field actually holds an error message key rather that the actual error message. When conflicts are encountered during approval attempt, the field holds the"CONFLICT_EXISTS" key.
- *Process id* uniquely identifies the batch run. The field is populated for records that are not in New status.
- *Clearance display id* is populated only upon 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 structure is similar to that of `RPM_PROMO_COMP_SIMPLE` but with the limitation that no exceptions are allowed.

Other than carrying data payload, the table holds fields that facilitate processing.

- *Auto approve indicator* defines whether the processing batch should attempt to approve the promotion after the data has been successfully imported
- *Status* defines the current state of the data in the staging table, as a result of processing. This status should not be confused with the status of the price event in RPM, even though there are some correlations. Status codes are as follows:
  - N (New): New status indicates that the data hasn't been processed yet
  - W (Worksheet): Worksheet status indicates that the data has been processed and has been imported successfully into RPM. It also indicates that at the time of import, the approval wasn't required
  - A (Approved): Approved status indicates that the data has been imported and approved without any conflicts.

- E (Error): Error status indicates that the data has been processed, but import has failed due to invalid data. The data administrator can correct the data and rerun the job.
- F (Failed). Failed status indicates that the data has been imported successfully, but some conflicts have been encountered. The data administrator can correct the data (for example, change the effective date of the event to eliminate the conflict) and rerun the job.

> **Note:** Initial data should be created in New status. The rest of the statuses are the result of data lifecycle. Records in any status (other than Approved) are eligible for processing by the batch.

- *Error message* holds the message for the first error encountered while importing the data. The field actually holds an error message key rather that the actual error message.
- *Process id* uniquely identifies the batch run. The field is populated for records that are not in New status.

Main Steps Taken by the Batch

1. Pricing events are generated. (import pricing events). This step loads data from the staging table (actual table depends on the event type specified as a batch argument). The batch validates the data based on RPM validity rules. If at least a single field for a record is not valid, the record is rejected. The first encountered error isreported (ERROR_MESSAGE column in the staging table will be populated), and the status is set to ERROR. The records with multiple incorrect data fields are processed multiple times unless corrected by the data administrator all at once. If no pricing events are generated the batch terminates at this stage. All records in the staging table that match the run argument criteria are processed at once. At the same time all records are independent in that data errors encountered on one record will not impact processing of other records.
2. If at least a single pricing event is generated, the batch proceeds with an optional approval step. For the batch to attempt approval of a record, the data administrator populating the staging data sets the auto approval flag on the record to ON (AUTO_APPRVE_IND should be set to 1). In this case, the batch attempts to approve the price event. This step involves conflict checking. This process can take a long time depending on the volume of data. It will end only when the approval process reports 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.
3. If auto approval is not requested or the approval process fails, the data is left in appropriate RPM table in Worksheet status. If approval is successful, the data is in Approved status. The status on the staging table after the approval step is either Approved or Failed, depending on how the process terminates.
4. The conflict checking logic does not purge intermediate data to allow the batch to inquire on the state of the conflict checking process. After it is determined that the CC logic is complete the batch purges CC intermediate data.

5. The final step for the batch process is to generate a report. The report gives the system administrator statistics on the batch run. The following information is provided:
   - Initial status the batch processed
   - Event type
   - Number of records imported
   - Number of records that required approval
   - Number of records successfully approved

Assumptions and Scheduling Notes

- A single instance of the batch runs at a time, so a single event type processes at a time.
- Approved pricing events on the staging tables cannot 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

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

InjectorPriceEventBatch Batch—Rollback and Reprocessing

If there is a mistake (such as a wrong date or retail) in the data file and bulk numbers of price events are created with the data, it is necessary to roll back all data to reprocess the file with the correct values.

The following are the steps to change a price change or clearance and promotion from Approved to Worksheet status:

1. Set up the data in the appropriate staging table with item/locations, status of N and auto_approve = 1.
2. Run the price injector batch, status parameter of N. Price events are created in the user interface and the staging table is updated with status of A (Approved).
3. To set the approved events back to Worksheet status, leave the same item/locations in the table from Step 1. Run the price injector batch with a status parameter of A. This means that all of the price events should be executed with status of A; doing so sets the price event back to Worksheet status.
4. Verify that the price events are set back to Worksheet status in the staging table and the user interface.

ItemLocDeleteBatch Batch

The ItemLocDeleteBatch program handles RMS deletions of item locations. When RMS deletes an item location, RPM now removes the Item/Location rows from the RPM_FUTURE_RETAIL table so that pricing events are no longer published out of RPM.

These item location deletions can be processed through either of two methods:

- The RMS_TABLE_RPM_ITL_AIR trigger
- A RIB message

These two options work the same as the batch and RIB modes for the New Item Location batch introduced in release 11.0.8.3. Documentation for switching between batch or RIB modes is found in the Release Notes for 11.0.8.3.

In the batch mode, the RPM_STAGE_DELETED_ITEM_LOC table is populated by the trigger RMS_TABLE_RPM_ITL_AIR. In RIB mode, the RPM_STAGE_DELETED_ITEM_LOC table is populated by subscribing to the itemlocmod and itemlocdel messages from RMS.

Usage

The following command runs the ItemLocDeleteBatch job:

`ItemLocDeleteBatch.sh` *`userid password`*

The first argument is the user ID and the second argument is the password.

Follow these steps to prepare to use this batch when the RIB is turned off:

1. Delete existing records from the table RPM_STAGE_DELETED_ITEM_LOC.
2. Enable the new trigger RMS_TABLE_RPM_ITL_AIR on table ITEM_LOC.
3. Stop the listener for Item/Location Creation RIB messages, as follows:
4. Log in to the Websphere administration console.
5. Select the RIBforRPM server from Servers > Application Servers.
6. Click the Message Listener Service link.
7. Click the Listener Ports link.
8. Select the check box next to ItemLocToRPMPort.
9. Click Stop. The listener is now stopped.
10. Click the ItemLocToRPMPort link to configure the port.
11. Select "Stopped" from the Initial State combo box.
12. Click OK.
13. Restart the RIBforRPM application server and verify that ItemLocToRPMPort is stopped.
14. Delete JMS subscriber in Egate, as follows:
15. Log in to the Egate Schema Manager.
16. Click the JMS Administrator button in the toolbar.
17. Expand the item etItmLocFromRMS.
18. Right-click on the RPM subscriber (it should have "RPM" in its name) and select "delete subscriber."

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

Modify the filter in the rib.properties file for RIBforRPM as follows, for RPM to subscribe to itemlocmod and itemlocdel:

`tafr.types.filter.itemloc=ItemLocCre,ItemLocMod,ItemLocDel`

1. Start the listener for Item/Location Creation RIB messages, as follows:
2. Log in to the Websphere administration console.
3. Select the RIBforRPM server from Servers > Application Servers.
4. Click the Message Listener Service link.
5. Click the Listener Ports link.
6. Select the check box next to ItemLocToRPMPort.
7. Click Start. The listener is now started.
8. Click the ItemLocToRPMPort link to configure the port.
9. Select "Started" from the Initial State combo box.
10. Click OK.
11. Restart the RIBforRPM app server and verify that ItemLocToRPMPort is started.

12. Create a JMS subscriber in Egate.

    Note: No action is needed.

13. Disable the new trigger RMS_TABLE_RPM_ITL_AIR on ITEM_LOC.
14. Run the new batch (ItemLocDeleteBatch.sh) one time to process any records remaining in the RPM_STAGE_DELETED_ITEM_LOC table.

Scheduling Notes

This batch can be run ad hoc.

itemReclassBatch Batch Design

Overview

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.

Usage

The following command runs the ItemReclassBatch job:

```
ItemReclassBatch userid password
```

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

Detail

The batch calls the package RPM_ITEM_RECLASS_SQL.RECLASS_FUTURE_RETAIL. This package looks for items in the RPM_ITEM_MODIFICATION table and updates the table RPM_FUTURE_RETAIL with the new department/class/subclass. The package then subsequently deletes all the records in the RPM_ITEM_MODIFICATION table.

Assumptions and Scheduling Notes

The RPM_ITEM_MODIFICATION table must have been already populated with the reclassed items by the ItemModification injector.

Primary Tables Involved

- RPM_ITEM_MODIFICATION
- RPM_FUTURE_RETAIL

Threading

itemReclassBatch is not threaded.

PL/SQL Interface Point

- Package: RPM_ITEM_RECLASS

LocationMoveBatch Batch Design

Overview

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

Usage

The following command runs the LocationMoveBatch job:

```
LocationMoveBatch [<username> <password>][max_retry]
```

Where the first argument is the user id, and the second argument is the password. Username and password are optional arguments. The third argument (max_retry) also is optional and specifies the maximum retry count for the batch.

To set up LocationMoveBatch for optional username and password, the following lines should be "commented out" in the launchRpmBatch.sh file:

```
if [ -z "$BATCH_USERNAME" ] ; then
    echo "You must specify a username."
    exit
fi

if [ -z "$BATCH_PASSWORD" ] ; then
    echo "You must specify a password."
    exit
fi
```

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 affected 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 affected 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.

Assumptions and Scheduling Notes

LocationMoveBatch must run before the following programs:

- PriceEventExecutionBatch
- MerchExtractKickOffBatch

Primary Tables Involved

RPM_FUTURE_RETAIL

Threading

LocationMoveBatch 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>`*

The optional mode argument can be used to split the processing into three components: pre-process, process, and post-process. The valid values for the mode argument are PRE, PROCESS, POST, and ALL. ALL is the default value for the mode argument when no value is provided.

The program is split 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 efficient manner. The splitting of the worksheet creation section ensures that a worksheet is not reprocessed in the case of a failure in a different worksheet. The splitting of a post process helps to avoid locking issues.

Detail

1. Setup (for modes ALL and PRE): Clean up expired worksheets and prepare for creation of new worksheets.
   a. Delete worksheets that are at the end of their review period.
   b. Get a list of all strategies that need to be processed today. Create copies of the strategies as needed.
   c. Determine what strategies need to be grouped together based on the RPM_DEPT_AGGREGATION. WORKSHEET_LEVEL.
   d. Stage 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 and allows the program to access large tables efficiently as possible.
2. Worksheet Creation (for modes ALL and PROCESS):
   a. Start threads based on the values in RPM_BATCH_CONTRL for MerchExtractKickOffBatch.java.
   b. 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.

c. For each RPM_WORKSHEET_DETAIL record created, do the following:
   - Use the price strategy to propose a retail value.
   - Apply candidate rules.
   - Apply price guides.

   The following are reasons why an item/location does not appear 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 market basket codes vary across locations in a zone.
   - The item 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 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.

3. Post process (for 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 replaces for the Item/Location Creation RIB message, the following steps should be followed in preparation for using this batch in place of the RIB:

1. Delete existing records from table RPM_STAGE_ITEM_LOC
2. Enable the new trigger RMS_TABLE_RPM_ITL_AIUDR in table ITEM_LOC.
3. Stop the MDB for Item/Location Creation RIB messages:
   a. Log in to Oracle Application Server Enterprise Manager.
   b. Expand the ribrpm-oc4j-instance server from All Application Servers.
   c. Click the rib-rpm application link.
   d. Click the rib-rpmEJB link.

e. In the Message Driven Beans section, select ItemLocToRPM.
f. Click Yes on the confirmation screen.
g. Restart the ribrpm-oc4j-instance and verify that ItemLocToRPM is stopped.

4. Delete JMS subscriber in Egate
   a. Login to the Egate Schema Manager
   b. Click the "JMS Administrator" button in the toolbar.
   c. Expand item "etItmLocFromRMS"
   d. Right click on the RPM subscriber (it should have "RPM" in its name) and select "delete subscriber"
5. 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:

1. Start the MDB for Item/Location Creation RIB messages:
   a. Log in to Oracle Application Server Enterprise Manager.
   b. Expand the ribrpm-oc4j-instance server from All Application Servers.
   c. Click the rib-rpm application link
   d. Click the rib-rpmEJB link
   e. In the Message Driven Beans section, select ItemLocToRPM.
   f. Restart the ribrpm-oc4j-instance and verify that ItemLocToRPM is up.
2. Create JMS subscriber in Egate (no action is needed).
3. Disable the new trigger RMS_TABLE_RPM_ITL_AIUDR on ITEM_LOC.
4. Run the new batch (NewItemLocBatch.sh) one time to process any records remaining in RPM_STAGE_ITEM_LOC.

To set up NewItemLocBatch for optional username and password, the following lines should be "commented out" in the launchRpmBatch.sh file:

```
if [ -z "$BATCH_USERNAME" ] ; then
    echo "You must specify a username."
    exit
fi

if [ -z "$BATCH_PASSWORD" ] ; then
    echo "You must specify a password."
    exit
fi
```

Detail

NewItemLocBatch 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

NewItemLocBatch 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

NewItemLocBatch 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/location combination.

Bulk Conflict Checking

NewItemLocBatch utilizes bulk conflict checking framework inside of RPM. Each thread that is spawned by the RPM batch threading framework (threaded by item/location) 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:

NewItemLocBatch 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, the program should be rerun with status E in an attempt to get the item/location 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 username and password are required arguments.

Additional Notes

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

Detail

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)

PriceChangeAutoApproveResultsPurgeBatch Batch Design

Overview

The PriceChangeAutoApproveResultsPurgeBatch program deletes old error message from the price change auto approve batch program.

Usage

The following command runs the PriceChangeAutoApproveResultsPurgeBatch job:

```
PriceChangeAutoApproveResultsPurgeBatch userid password
```

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

Detail

The PriceChangeAutoApproveResultsPurgeBatch program deletes price change auto approve errors. These errors are generated when the WorksheetAutoApproveBatch cannot approve a price change that it has created. The price change auto approve errors are deleted based on the effective dates of the price changes associated with the error. The price change auto approve errors are delete when the effective date of the price changes attempted to be approved is less than or equal to the vdate.

Assumptions and Scheduling Notes

PriceChangeAutoApproveResultsPurgeBatch can be run ad hoc.

Primary Tables Involved

- RPM_MAINT_MARGIN_ERR
- RPM_MAINT_MARGIN_ERR_DTL

Threading

PriceChangeAutoApproveResultsPurgeBatch program is not threaded.

PriceChangePurgeBatch Batch Design

Overview

The PriceChangePurgeBatch program deletes past price changes.

Usage

The following command runs the PriceChangePurgeBatch job:

```
PriceChangePurgeBatch userid password
```

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

Detail

The PriceChangePurgeBatch program deletes price changes that have an effective date that is less than the vdate.

Assumptions and Scheduling Notes

PriceChangePurgeBatch can be run ad hoc.

Primary Tables Involved

- RPM_PRICE_CHANGE

Threading

PriceChangePurgeBatch is not threaded.

PriceChangePurgeWorkspaceBatch Batch Design

Overview

The PriceChangePurgeWorkspaceBatch program deletes abandoned price change workspace records.

Usage

The following command runs the PriceChangePurgeWorkspaceBatch job:

```
PriceChangePurgeWorkspaceBatch userid password
```

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

Detail

When users access the price change dialogue, records are created in workspace tables. These records are typically removed when the user exits the price change dialogue. However, it is possible that the workspace records may be abandoned. When this action occurs, the PriceChangePurgeWorkspaceBatch deletes them. The PriceChangePurgeWorkspaceBatch deletes records in the workspace table that are over n days old, where n is a system defined number of days.

Assumptions and Scheduling Notes

PriceChangePurgeWorkspaceBatch can be run ad hoc.

Primary Tables Involved

- RPM_PRICE_WORKSPACE
- RPM_PRICE_WORKSPACE_DETAIL

Threading

PriceChangePurgeWorkspaceBatch is not threaded.

Price Event Execution Batch Processes

Overview

The price event execution batch processes perform the necessary work to start (regular price change, clearance price change, promotions) and end (price change, promotions) pricing events.

Executing price events require running three batch programs. These are:

- PriceEventExecutionBatch.java identifies the events that need to be executed and stages the affected item-locations for the next batch to process. If this batch fails to process a particular price event, that event will remain in "approved" status and the next-day batch run is guaranteed to pick up this failed price event for re-processing.

- PriceEventExecutionRMSBatch.java processes the item-locations affected by the price events being executed RMS. If this batch fails to process a particular item-location for one or more price events, the affected events will be in "executed" status and the item-locations that failed to process will remain staged in RPM_EVENT_ITEMLOC. These item-locations will be picked up again by the next-day batch run.
- PriceEventExecutionDealsBatch.java processes the deals affected by the price events being executed. If this batch fails to process a particular item-location deal for one or more price events, the affected events will be in "executed" status and their associated item-locations are posted in RMS ITEM_LOC and PRICE_HIST tables. However, the item-location deals that failed to process will remain in RPM_EVENT_ITEMLOC_DEALS and the next-day batch run is guaranteed to pick these up.

Usage

The following commands will need to be executed in order:

```
PriceEventExecutionBatch userid password
PriceEventExecutionRMSBatch userid password
PriceEventExecutionDealsBatch userid password
```

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

Detail

The batch programs process regular price changes, clearance price changes, and promotions events that are scheduled for the run date. Restartability features allow events missed in past runs of the batch to be picked up in later runs. When posting information in the ITEM_LOC and PRICE_HIST table, the batch process will respect the active dates of their associated price events.

- Promotions:
  - Promotions that are scheduled to start are activated. These include all approved promotions whose start dates are <= VDATE+1.
  - Promotions that are scheduled to end are completed. These include all active promotions whose end dates are <= VDATE.
- Clearances:
  - Clearance markdowns that are scheduled to take place are executed. These include all clearances whose effective dates are <= VDATE+1.
  - Clearances that are scheduled to be completed (reset) are completed.
- Regular price changes:
  - Regular price changes that are scheduled to take place are executed. These include all price changes whose effective dates are <= VDATE+1.

Assumptions and Scheduling Notes

The batch processes must run in the following order:

- PriceEventExecutionBatch
- PriceEventExecutionRMSBatch
- PriceEventExecutionDealsBatch

The previous three processes must run before the following programs:

- Storeadd (RMS)
- MerchExtractKickOffBatch

The following programs must run before the PriceEventExecution batch processes:

- Salstage (RMS)
- LocationMoveBatch

Primary tables involved

- RPM_PRICE_CHANGE
- RPM_CLEARANCE
- RPM_PROMO_COMP_DETAIL

RMS interface point

The PriceEventExecutionRMSBatch interfaces with the RMS price change subscription package RMSSUB_PRICE_CHANGE. All price change, clearance, and promotion prices will be passed along to this RMS package at the item location level and will be applied in RMS.

Threading

Two of the three batch programs involved in price event execution utilize concurrent processing. These are PriceEventExecutionBatch and PriceEventExecutionRMSBatch. The threading strategies for these two batch programs are different from each other.

PriceEventExecutionBatch is threaded by a variable number of pricing events to be executed (i.e., price changes, clearances, and promotions).

PriceEventExecutionRMSBatch is threaded by a variable number of item-locations affected by the pricing events to be executed.

PriceStrategyCalendarBatch Batch Design

Overview

The calendar expiration batch process (PriceStrategyCalendarBatch.java) maintains calendars assigned to price strategies.

Usage

The following command runs the PriceStrategyCalendarBatch job:

```
PriceStrategyCalendarBatch userid password
```

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

Detail

The batch looks at price strategies that have expired or suspended calendars.

If a strategy has new calendars setup, the batch replaces the strategies' current calendar with the new calendar.

If a strategy does not have a new calendars setup and the expired calendar has a replacement calendar specified, the batch replaces the strategies' current calendar with the current calendar's replacement calendar.

Assumptions and Scheduling Notes

PriceStrategyCalendarBatch must run before the following programs:

- PriceEventExecutionBatch,
- MerchExtractKickOffBatch

Primary Tables Involved

- RPM_STRATEGY
- RPM_CALENDAR
- RPM_CALENDAR_PERIOD

Threading

PriceStrategyCalendarBatch is not threaded.

PromotionPriceChangePublishBatch Batch Design

Overview

The PromotionPriceChangePublishBatch program formats and stages output of promotion price change price events.

The corresponding promotionPriceChangePublishExport shell script produces a pipe ('|') delimited flat-file export based on the output of the PromotionPriceChangePublishBatch.

Usage

The following command runs the PromotionPriceChangePublishBatch job:

```
PromotionPriceChangePublishBatch userid password
```

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

The following command runs the promotionPriceChangePublishExport job:

```
promotionPriceChangePublishExport.sh database-connect-string path
```

Where the first argument is the database connect string (user/pwd@database), and the second argument is the path where the file should be written. The path is optional. If it is not supplied, the path ../output will be used.

Detail

The batch looks for price events in the RPM_PRICE_EVENT_PAYLOAD table with a RIB_FAMILY of "PRMPRCCHG" and distributes those events to multiple threads based on the settings in the RPM_BATCH_CONTROL table. Each thread reads in its set of promotion price change events from tables RPM_PRICE_EVENT_PAYLOAD and the related promotion payload tables (see below) and generates output in RPM_PRICE_PUBLISH_DATA.

PurgePayloadsBatch should be run to delete records in the payload tables.

A single flat-file (PRMPC_<timestamp>.pub) is created, where <timestamp> is the current system time stamp.

Input Tables

RPM_PRICE_EVENT_PAYLOAD

RPM_PROMO_DTL_PAYLOAD

RPM_PROMO_DTL_SMP_PAYLOAD

RPM_PROMO_DTL_THR_PAYLOAD

RPM_THRESHOLD_DTL_PAYLOAD

RPM_PROMO_DTL_BG_PAYLOAD

RPM_PROMO_DTL_BGI_PAYLOAD

Output File Record Types

FHEAD – REQUIRED: File identification, one line per file.
TSMPP – OPTIONAL: Simple Promotion (transaction head)
TSDTL – REQUIRED: Simple Promotion Detail (1 or more per TSMPL)
TTAIL – REQUIRED: Transaction tail (1 per promotion)
FSDEL – OPTIONAL: Simple Promotion Delete
TTHRP – OPTIONAL: Threshold Promotion (transaction head)
TTLVL – REQUIRED: Threshold Level (1 or more per TTHRP)
TTDTL – REQUIRED: Threshold Detail (1 or more per TTHRP)
TTAIL – REQUIRED: Transaction Tail (1 per promotion)
FTDEL – OPTIONAL: Threshold Promotion Delete
TBGTP – OPTIONAL: Buy/Get Promotion (transaction head)
TBITM – REQUIRED: Buy Item (1 or more per TBGTP)
TGITM – REQUIRED: Get Item (1 or more per TBGTP)
TLOCN – REQUIRED: Location (1 or more per TBGTP)
TTAIL – REQUIRED: Transaction tail (1 per promotion)
FBDEL – OPTIONAL: Buy/Get Promotion Delete
FTAIL – REQUIRED: End of file marker, one line per file.

Output File Layout

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record Descriptor	Char(5)	FHEAD	File head marker
	Line id	Number(10)	1	Unique line id
	File Type	Char(5)	REGPC	Regular Price Changes
	Export timestamp	Timestamp		System clock timestamp (YYYYMMDDHHMISS)
	Format Version	Char(5)	1.0	File Format Version
TSMPP	Record Descriptor	Char(5)	TSMPP	Simple Promotion (transaction head)
	Line id	Number(10)		Unique line id
	Event Type	Char(3)		"CRE" = Create, "MOD" = Modify
	Promotion Id	Number(10)		Promotion Id
	Promo Comp Id	Number(10)		Promotion Component Id
	Promo Comp Detl Id	Number(10)		Promotion Component Detail Id
	Start Date	Date		Start Date of Promotion Component Detail (DD-MMM-YY)
	End Date	Date		End Date of Promotion Component Detail (DD-MMM-YY)
	Promo Name	Char(160)		Promotion Header Name

Record Name	Field Name	Field Type	Default Value	Description
	Promo Desc	Char(640)		Promotion Header Description
	Promo Comp Name	Char(160)		Promotion Component Name
	Apply Order	Number(1)		Application Order of the Promotion
	Change Type	Char(1)		Retail Change Type ['A'mount Off or 'P'ercent Off or 'F'ixed Price or 'N'one]
	Change Value	Number(20,4)		Retail Change Value [Value of the promotion]
	Change Value UOM	Char(4)		Unit of Measure of the change value [if supplied]
TSDTL	Record Descriptor	Char(5)	TSDTL	Simple Promotion Detail
	Line id	Number(10)		Unique line id
	Item Id	Char(25)		Transaction Item Identifier
	Location Id	Number(10)		Org Node [Store or Warehouse] identifier
	Location Type	Char(1)		Org Node Type ['S'tore or 'W'arehouse]
	Selling Retail	Number(20,4)		Selling Retail with promotion applied
	Selling Retail UOM	Char(4)		Unit of Measure of Selling Retail
TTAIL	Record Descriptor	Char(5)	TTAIL	Transaction Tail
	Line id	Number(10)		Unique line id
FSDEL	Record Descriptor	Char(5)	FSDEL	Simple Promotion Delete (file detail)
	Line id	Number(10)		Unique line id
	Promo Comp Detail Id	Number(10)		Promotion Component Detail Id
	Item Id	Char(25)		Transaction Item Identifier
	Location Id	Number(10)		Org Node [Store or Warehouse] identifier
	Location Type	Char(1)		Org Node Type [Store or Warehouse]
TTHRP	Record Descriptor	Char(5)	TTHRP	Threshold Promotion (transaction head)
	Line id	Number(10)		Unique line id
	Event Type	Char(3)		"CRE" = Create, "MOD" = Modify
	Promotion Id	Number(10)		Promotion Id
	Promo Comp Id	Number(10)		Promotion Component Id

Record Name	Field Name	Field Type	Default Value	Description
	Promo Comp Detl Id	Number(10)		Promotion Component Detail Id
	Start Date	Date		Start Date of Promotion Component Detail (DD-MMM-YY)
	End Date	Date		End Date of Promotion Component Detail (DD-MMM-YY)
	Promo Name	Char(160)		Promotion Header Name
	Promo Desc	Char(640)		Promotion Header Description
	Promo Comp Name	Char(160)		Promotion Component Name
	Apply Order	Number(1)		Application Order of the Promotion
	Threshold Id	Number(6)		Threshold Id
	Threshold Name	Char(120)		Threshold Name
	Threshold Qual Type	Number(1)		Threshold Qualification Type [0=Item, 1=Threshold]
	Threshold Type	Char(1)		Threshold Type ['Q'uantity or 'A'mount]
	Change Type	Char(1)		Change Type ['A'mount Off or 'P'ercent Off or 'F'ixed Price]
TTLVL	Record Descriptor	Char(5)	TTLVL	Threshold Level Detail
	Line id	Number(10)		Unique line id
	Threshold Value	Number(20,4)		Threshold Value [Amount or Quantity] needed to hit threshold
	Change Value	Number(20,4)		Retail Change Value [Value of the promotion threshold detail]
	Change Value UOM	Char(4)		Unit of Measure of the change value [if supplied]
TTDTL	Record Descriptor	Char(5)	TTDTL	Threshold Detail
	Line id	Number(10)		Unique line id
	Item Id	Char(25)		Transaction Item Identifier
	Location Id	Number(10)		Org Node [Store or Warehouse] identifier
	Location Type	Char(1)		Org Node Type ['S'tore or 'W'arehouse]
TTAIL	Record Descriptor	Char(5)	TTAIL	Transaction Tail
	Line id	Number(10)		Unique line id
FTDEL	Record Descriptor	Char(5)	FTDEL	Threshold Promotion Delete (file detail)

Record Name	Field Name	Field Type	Default Value	Description
	Line id	Number(10)		Unique line id
	Promo Comp Detail Id	Number(10)		Promotion Component Detail Id
	Item Id	Char(25)		Transaction Item Identifier
	Location Id	Number(10)		Org Node [Store or Warehouse] identifier
	Location Type	Char(1)		Org Node Type ['S'tore or 'W'arehouse]
TBGTP	Record Descriptor	Char(5)	TBGTP	Buy/Get Promotion (transaction head)
	Line id	Number(10)		Unique line id
	Event Type	Char(3)		"CRE" = Create, "MOD" = Modify
	Promotion Id	Number(10)		Promotion Id
	Promo Comp Id	Number(10)		Promotion Component Id
	Promo Comp Detl Id	Number(10)		Promotion Component Detail Id
	Start Date	Date		Start Date of Promotion Component Detail (DD-MMM-YY)
	End Date	Date		End Date of Promotion Component Detail (DD-MMM-YY)
	Promo Name	Char(160)		Promotion Header Name
	Promo Desc	Char(640)		Promotion Header Description
	Promo Comp Name	Char(160)		Promotion Component Name
	Apply Order	Number(1)		Application Order of the Promotion
	All Indicator	Number(1)		Buy Get all indicator [1=buy all, 0=any]
	Buy Quantity	Number(20,4)		Quantity needed for buy – get
	Change Type	Char(1)		Retail Change Type ['A'mount Off or 'P'ercent Off or 'F'ixed Price]
	Change Value	Number(20,4)		Retail Change Value [Value of the promotion]
	Change Value UOM	Char(4)		Unit of Measure of the change value [if supplied]
TBITM	Record Descriptor	Char(5)	TBITM	Buy Item
	Line id	Number(10)		Unique line id
	Buy Item Id	Char(25)		Transaction Item Identifier for buy item
TGITM	Record Descriptor	Char(5)	TGITM	Get Item

Record Name	Field Name	Field Type	Default Value	Description
	Line id	Number(10)		Unique line id
	Get Item Id	Char(25)		Transaction Item Identifier for get item
TLOCN	Record Descriptor	Char(5)	TLOCN	Location
	Line id	Number(10)		Unique line id
	Location Id	Number(10)		Org Node [Store or Warehouse] identifier
	Location Type	Char(1)		Org Node Type ['S'tore or 'W'arehouse]
TTAIL	Record Descriptor	Char(5)	TTAIL	Transaction Tail
	Line id	Number(10)		Unique line id
FBDEL	Record Descriptor	Char(5)	FBDEL	Buy/Get Promotion Delete (file detail)
	Line id	Number(10)		Unique line id
	Promo Comp Detail Id	Number(10)		Promotion Component Detail Id
	Item Id	Char(25)		Transaction Item Identifier
	Location Id	Number(10)		Org Node [Store or Warehouse] identifier
	Location Type	Char(1)		Org Node Type ['S'tore or 'W'arehouse]
FTAIL	Record Descriptor	Char(5)	FTAIL	File tail marker
	Line id	Number(10)		Unique line id
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

Assumptions and scheduling notes

PromotionPriceChangePublishBatch should be run after WorksheetAutoApproveBatch.

PromotionPriceChangePublishExport should be executed as follows:

- After every successful run of PromotionPriceChangePublishBatch.
- Before PurgePayloadsBatch.

Primary tables involved

- RPM_PRICE_EVENT_PAYLOAD
- RPM_PROMO_DTL_PAYLOAD
- RPM_PROMO_DTL_SMP_PAYLOAD
- RPM_PROMO_DTL_THR_PAYLOAD
- RPM_THRESHOLD_DTL_PAYLOAD
- RPM_PROMO_DTL_BG_PAYLOAD
- RPM_PROMO_DTL_BGI_PAYLOAD

Threading

PromotionPriceChangePublishBatch is threaded. The LUW is a single rpm_price_event_payload record. (Buy/Get promotions will not be split across threads, but Simple and Threshold promotions may be.)

Configuration

The following property must be set in rpm.properties on the application server when using this batch program to publish promotion price events:

```
delete_staged_rib_payloads=false
```

PromotionPurgeBatch Batch Design

Overview

The PromotionPurgeBatch program deletes old and rejected promotions.

Usage

The following command runs the PromotionPurgeBatch job:

```
PromotionPurgeBatch userid password
```

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

Detail

PromotionPurgeBatch deletes promotions based on two system options. RPM_SYSTEM_OPTIONS.PROMOTION_HIST_MONTHS and RPM_SYSTEM_OPTIONS.REJECT_HOLD_DAYS_PROMO. RPM_SYSTEM_OPTIONS.PROMOTION_HIST_MONTHS controls how long non-rejected promotions are held. RPM_SYSTEM_OPTIONS.REJECT_HOLD_DAYS_PROMO controls how long rejected promotions are held.

Assumptions and Scheduling Notes

PromotionPurgeBatch can be run ad hoc.

Primary Tables Involved

- RPM_PROMO_EVENT
- RPM_PROMO
- RPM_PROMO_DEAL_LINK
- RPM_PROMO_COMP
- RPM_PROMO_COMP_THRESH_LINK
- RPM_PROMO_DEAL_COMP_LINK
- RPM_PROMO_COMP_DETAIL
- RPM_PROMO_COMP_SIMPLE
- RPM_PROMO_COMP_THRESHOLD
- RPM_PROMO_COMP_BUY_GET
- RPM_PROMO_BUY_GET_ZONE
- RPM_PROMO_BUY_ITEM
- RPM_PROMO_GET_ITEM
- RPM_PENDING_DEAL
- RPM_PENDING_DEAL_DETAIL

Threading

PromotionPurgeBatch is threaded.

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

PurgeExpiredExecutedOrApprovedClearancesBatch Batch Design

Overview

The PurgeExpiredExecutedOrApprovedClearancesBatch program deletes expired clearances in 'Executed' or 'Approved' statuses.

Usage

The following command runs the PurgeExpiredExecutedOrApprovedClearancesBatch job:

```
PurgeExpiredExecutedOrApprovedClearancesBatch userid password
```

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

Detail

The PurgeExpiredExecutedOrApprovedClearancesBatch deletes clearances that meet the following criteria:

- Clearance effective date is older than the CLEARANCE_HIST_MONTHS system option.
- Clearance is on a valid future retail timeline (RPM_FUTURE_RETAIL).
- Clearance is in an Approved or Executed status.

Assumptions and Scheduling Notes

PurgeExpiredExecutedOrApprovedClearancesBatch can be run ad hoc.

Primary Tables Involved

- RPM_CLEARANCE

Threading

PurgeExpiredExecutedOrApprovedClearancesBatch is not threaded.

PurgeLocationMovesBatch Batch Design

Overview

The PurgeLocationMovesBatch program deletes old expired and executed zone location moves.

Usage

The following command runs the PurgeLocationMovesBatch job:

```
PurgeLocationMovesBatch userid password
```

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

Detail

The PurgeLocationMovesBatch program deletes location moves based on their effective date. Location moves are purged regardless whether or not they have been executed. Location moves are purged when their effective date is RPM_SYSTEM_OPTIONS. LOCATION_MOVE_PURGE_DAYS days in the past.

Assumptions and Scheduling Notes

PurgeLocationMovesBatch can be run ad hoc.

Primary Tables Involved

- RPM_LOCATION_MOVE
- RPM_LOC_MOVE_PROMO_ERROR
- RPM_LOC_MOVE_PRC_STRT_ERR
- RPM_LOC_MOVE_PRC_CHNG_EX
- RPM_LOC_MOVE_CLEARANCE_EX
- RPM_LOC_MOVE_PROMO_COMP_DTL_EX

Threading

PurgeLocationMovesBatch is not threaded.

PurgePayloadsBatch Batch Design

Overview

The PurgePayloadsBatch program purges entries related to price events from the RPM_*PAYLOAD tables.

Usage

The following command runs the PurgePayloadsBatch job:

```
purgePayloadsBatch.sh<userid/pwd@database><publish-status><log-path>
```

Where the first argument is the database connection string: username, password and database information.

The second argument is the publish status of the price event record. Valid values are 1, 2, and 3, as described in the following table:

<publish-status>	Description
1	This setting results in purged price events that have been exported by RPMtoORPOSPublishExport.sh
2	This setting results in purged price events that have been exported by any one of the following: ▪ regularPriceChangePublishExport.sh ▪ clearancePriceChangePublishExport.sh ▪ promotionPriceChangePublishExport.sh
3	This setting results in purged price events that have been exported by RPMtoORPOSPublishExport.sh AND by any one of the following: ▪ regularPriceChangePublishExport.sh ▪ clearancePriceChangePublishExport.sh ▪ promotionPriceChangePublishExport.sh

The third argument, <log-path>, specifies the log path for purgePayloadsBatch.

Detail

The PurgePayloadsBAtch purges the price vents from the payload tables based on the <publish-status> argument. Publish status is stored in the PUBLISH_STATUS column of the RPM_PRICE_EVENT_PAYLOAD table. Valid values are 0, 1, 2, and 3, as described in the following table:

PUBLISH_STATUS	Description
0	This is the default value for the field. The PurgePayloadsBatch job will not purge event records with PUBLISH_STATUS=0. The error message, "You must specify publish status: 1/2/3", displays if the user attempts to run the batch with <publish-status>=0.
1	Price event records with PUBLISH_STATUS=1 have been exported by RPMtoORPOSPublishExport.sh
2	Price event records with PUBLISH_STATUS=2 have been exported by any one of the following: ▪ regularPriceChangePublishExport.sh ▪ clearancePriceChangePublishExport.sh ▪ promotionPriceChangePublishExport.sh
3	Price event records with PUBLISH_STATUS=3 have been exported by RPMtoORPOSPublishExport.sh AND by any one of the following: ▪ regularPriceChangePublishExport.sh ▪ clearancePriceChangePublishExport.sh ▪ promotionPriceChangePublishExport.sh

Assumptions and Scheduling Notes

PurgePayloadsBAtch should be run after the successful completion of the following batch jobs:

- RPMtoORPOSPublishExport.sh
- regularPriceChangePublishExport.sh
- clearancePriceChangePublishExport.sh
- promotionPriceChangePublishExport.sh

Primary Tables Involved

RPM_PRICE_CHG_PAYLOAD
RPM_CLEARANCE_PAYLOAD
RPM_PROMO_DTL_PAYLOAD
RPM_PROMO_DTL_SMP_PAYLOAD
RPM_PROMO_DTL_THR_PAYLOAD
RPM_THRESHOLD_DTL_BG_PAYLOAD
RPM_PROMO_DTL_BG_PAYLOAD
RPM_PROMO_DTL_BGI_PAYLOAD
RPM_PRICE_EVENT_PAYLOAD

Threading

PurchPayloadsBatch is not threaded.

PurgeUnusedAndAbandonedClearancesBatch Batch Design

Overview

The PurgeUnusedAndAbandonedClearancesBatch program deletes unused and rejected clearances.

Usage

The following command runs the PurgeUnusedAndAbandonedClearancesBatch job:

```
PurgeUnusedAndAbandonedClearancesBatch userid password
```

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

Detail

The PurgeUnusedAndAbandonedClearancesBatch program deletes clearances from the RPM_CLEARANCE table that meet one of the following three criteria:

- Clearance effective date is older than the CLEARANCE_HIST_MONTHS system option.
- Clearance is in 'Worksheet' or 'Submitted' status.

Or

- Clearance effective date is older than the CLEARANCE_HIST_MONTHS system option.
- Clearance is in 'Execute' or 'Approved' status.
- Clearance is not on a future retail timeline

Or

- Clearance effective date is older than the REJECT_HOLD_DAYS_PC_CLEAR system option.
- Clearance is in 'Rejected' status.

Assumptions and Scheduling Notes

PurgeUnusedAndAbandonedClearancesBatch can be run ad hoc.

Primary Tables Involved

- RPM_CLEARANCE

Threading

PurgeUnusedAndAbandonedClearancesBatch is not threaded.

RefreshPosDataBatch Batch Design

Overview

The RefreshPosDataBatch batch is intended to provide the Point Of Sale (POS) application with data refresh capability. This batch generates all pricing information (current and future) for a store. The information is then sent to POS for the purpose of refreshing POS systems data.

Usage

The following command runs the RefreshPosDataBatch job

```
RefreshPosDataBatch <username> <password> <location>[date (YYYYMMdd)]
```

Where the first argument is the username. The second argument is the password, and the third argument is the store location id. The fourth argument is the action date from which the pricing information data is required. The action date must be in the format YYYYMMdd. If the action date is not provided, the field will default to VDATE+1.

Detail

The RefreshPosDataBatch batch deletes the contents of the payload tables listed below. It then looks for price events pertaining to the specified store id in RPM_FUTURE_RETAIL table and populates the payload tables with price event information, starting from the specified action date till the current day. If the action date is not specified, the program populates the payload tables with data starting from V_DATE+1. If the Warehouse id

needs to be specified as a location, then the system option, Recognize WHs as Location, must be set in System Options.

Assumptions and Scheduling Notes

RefreshPosDataBatch can be run ad hoc.

Primary Tables Involved

RPM_FUTURE_RETAIL
RPM_PRICE_EVENT_PAYLOAD
RPM_PRICE_CHG_PAYLOAD
RPM_CLEARANCE_PAYLOAD
RPM_PROMO_DTL_SMP_PAYLOAD
RPM_PROMO_DTL_PAYLOAD
RPM_PROMO_DTL_BG_PAYLOAD
RPM_PROMO_DTL_BGI_PAYLOAD
RPM_PROMO_DTL_THR_PAYLOAD
RPM_THRESHOLD_DTL_PAYLOAD

Output

There is no separate output. The data from the RPM_FUTURE_RETAIL populates the tables above.

Threading

The program is threaded based on the department count.

RegularPriceChangePublishBatch Batch Design

Overview

The RegularPriceChangePublishBatch program formats and stages output of regular price change price events.

The corresponding regularPriceChangePublishExport shell script produces a pipe ('|') delimited flat-file export based on the output of RegularPriceChangePublishBatch.

Usage

The following command runs the RegularPriceChangePublishBatch job:

```
RegularPriceChangePublishBatch userid password
```

Where the first argument is the RPM user id.

The second argument is the password.

The following command runs the regularPriceChangePublishExport job:

```
regularPriceChangePublishExport.sh database-connect-string path
```

Where the first argument is the database connect string (user/pwd@database)

The second argument is the path where the file should be written. The path is optional, and if it is not supplied, the path ../output will be used.

Detail

This batch job looks for price events in the RPM_PRICE_EVENT_PAYLOAD table with a RIB_FAMILY of REGPRCCHG and distributes those events to multiple threads based on the settings in the RPM_BATCH_CONTROL table. Each thread reads in its set of regular price change events from tables RPM_PRICE_EVENT_PAYLOAD and RPM_PRICE_CHG_PAYLOAD and generates output in RPM_PRICE_PUBLISH_DATA.

PurgePayloadsBatch should be run to delete records in the payload tables.

A single flat-file (REGPC_<timestamp>.pub) is created, where <timestamp> is the current system time stamp.

Output File

FHEAD – REQUIRED: File identification, one line per file.

FDETL – OPTIONAL: Price Change Event (Create or Modify)

FDELE – OPTIONAL: Price Change Event (Delete)

FTAIL – REQUIRED: End of file marker, one line per file.

Output File Layout

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record Descriptor	Char(5)	FHEAD	File head marker
	Line id	Number(10)	1	Unique line id
	File Type	Char(5)	REGPC	Regular Price Changes
	Export timestamp	Timestamp		System clock timestamp (YYYYMMDDHHMISS)
	Format Version	Char(5)	1.0	File Format Version
FDETL	Record Descriptor	Char(5)	FDETL	File Detail Marker (1 per price change create or modify)
	Line id	Number(10)		Unique line id
	Event Type	Char(3)		"CRE" = Create, "MOD" = Modify
	Id	Number(15)		Price Change identifier
	Item	Char(25)		Item identifier
	Location	Number(10)		Location identifier
	Location Type	Char(1)		'S' = Store, 'W' = Warehouse
	Effective Date	Date		Effective Date of price change (DD-MMM-YY)
	Selling Unit Change Ind	Number(1)		Did selling unit retail change with this price event (0 = no change, 1 = changed)
	Selling Retail	Number(20,4)		Selling retail with price change applied
	Selling Retail UOM	Char(4)		Selling retail unit of measure

Record Name	Field Name	Field Type	Default Value	Description
	Selling Retail Currency	Char(3)		Selling retail currency
	Multi-Unit Change Ind	Number(1)		Did multi unit retail change with this price event (0 = no change, 1 = changed)
	Multi-Units	Number(12,4)		Number Multi Units
	Multi-Unit Retail	Number(20,4)		Multi Unit Retail
	Multi-Unit UOM	Char(4)		Multi Unit Retail Unit Of Measure
	Multi-Unit Currency	Char(3)		Multi Unit Retail Currency
FDELE	Record Descriptor	Char(5)	FDELE	File Detail Delete Marker (1 per price change delete)
	Line id	Number(10)		Unique line id
	Id	Number(15)		Price Change identifier
	Item	Char(25)		Item identifier
	Location	Number(10)		Location identifier
	Location Type	Char(1)		'S' = Store, 'W' = Warehouse
FTAIL	Record Descriptor	Char(5)	FTAIL	File tail marker
	Line id	Number(10)		Unique line id
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

Assumptions and Scheduling Notes

RegularPriceChangePublishBatch should be run after WorksheetAutoApproveBatch.

RegularPriceChangePublishExport should be executed as follows:

- After every successful run of RegularPriceChangePublishBatch.
- Before PurgePayloadsBatch.

Primary Tables Involved

- RPM_PRICE_EVENT_PAYLOAD
- RPM_PRICE_CHG_PAYLOAD

Threading

RegularPriceChangePublishBatch is threaded. The LUW is a single regular price change event.

Configuration

The following property must be set in rpm.properties on the application server when using this batch program to publish regular price events:

```
delete_staged_rib_payloads=false
```

RPMtoORPOSPublishBatch Batch Design

Overview

The RPMtoORPOSPublishBatch program formats and stages output of price events, such as PriceChange, Clearance and Promotion. The RPMtoORPOSPublishExport shell script produces an xml file based on the output of the RPMtoORPOSPublishBatch job.

Usage

The following command runs the RPMtoORPOSPublishBatch job:

```
RPMtoORPOSPublishBatch <database-connect-string> <log path> <error path>
```

Where the first argument is the RPM database-connect-string (user/pwd@database).

The second argument is the log path where the log file is written.

The third argument is the error path where the error file is written.

Detail

The batch takes data from the payload and price event tables, depending on the price event. Each price event has its own payload tables, as shown in the table below:

S.No.	Price Event	Related Tables
1	Regular Price Change	RPM_PRICE_CHG_PAYLOAD, RPM_PRICE_EVENT_PAYLOAD
2	Clearance	RPM_CLEARANCE_PAYLOAD, RPM_PRICE_EVENT_PAYLOAD
3	Simple Promotion	RPM_PROMO, RPM_PROMO_DTL_PAYLOAD, RPM_PROMO_DTL_SMP_PAYLOAD, RPM_PRICE_EVENT_PAYLOAD
4	Threshold Promotion	RPM_PROMO_DTL_PAYLOAD, RPM_THRESHOLD_DTL_PAYLOAD,RPM_PROMO_DTL_THR_PAYLOAD, RPM_PRICE_EVENT_PAYLOAD, RPM_PRICE_EVENT_PAYLOAD
5	Buy Get Promotion	RPM_PROMO_DTL_PAYLOAD, RPM_PROMO_DTL_BG_PAYLOAD, RPM_PROMO_BUY_ITEM, RPM_PRICE_EVENT_PAYLOAD RPM_PROMO_DTL_BGI_PAYLOAD

The data collected from all these payload tables are formatted and inserted in various tables, as shown in the table below:

S.No.	Price Event	Table Name
1	Price Change	RPM_ORPOS_PRICE_CHANGE_PUBLISH
2	Clearance	RPM_ORPOS_PRICE_CHANGE_PUBLISH
3	Simple Promo	RPM_ORPOS_SIMPLE_PROMOTION_PUBLISH

S.No.	Price Event	Table Name
4	Threshold Promo	RPM_ORPOS_COMPLEX_PROMOTION_PUBLISH
5	BuyGet Promo	RPM_ORPOS_COMPLEX_PROMOTION_PUBLISH

Output

There is no separate output. The data mentioned in the tables above are the output of the batch program.

Assumptions and Scheduling Notes

- RPMtoORPOSPublishBatch should be run after WorksheetAutoApproveBatch.
- RPMtoORPOSPublishBatch should be run before RPMto ORPOSPublishExport.

Primary Tables Involved

- RPM_ORPOS_PRICE_CHANGE_PUBLISH
- RPM_ORPOS_SIMPLE_PROMOTION_PUBLISH
- RPM_ORPOS_COMPLEX_PROMOTION_PUBLISH

Configuration

The following property must be set in rpm.properties on the application server when using this batch program to publish all price events:

```
delete_staged_rib_payloads=false
```

RPMtoORPOSPublishExport Batch Design

Overview

The RPMtoORPOSPublishExport program calls an SQL script, RPMtoORPOSSpoolMsg.SQL, that spools the data collected from various publish tables of price events (such as Price Change, Clearance, and Promotion). The spooled file is in xml format and complies with the given XSD standard.

Usage

The following command runs the RPMtoORPOSPublishExport job:

```
RPMtoORPOSPublishExport <database-connect-string> <Number of slots>  <log path>
<error path> <Export path>
```

Where the first argument is the RPM database-connect-string (user/pwd@database).

The second argument is the Number of slots, which denotes the number of threads to be executed in parallel.

The third argument is the log path where the log file is written.

The fourth argument is the error path where the error file is written.

The fifth argument is the Export path or Result path where the generated xml files are written. The export path is optional. If it is not supplied, the present working directory (or "pwd") will be used instead.

> **Note:** The number of slots variable must be greater than zero.

Note: The log and error path parameters must be a valid directory at the same level as PWD.

Detail

The batch looks for data in various publish tables. The batch invokes a SQL script by passing the store_id. In the SQL script, the spool filename is generated as follows: PricingExtract_<store_id>.xml. A file will be generated for all the stores, whether price events exist or not. Each price event possesses different structures and values that comply with the XSD standard.

OutputFile

The output file is an xml file generated on a per-store basis. The XML file should comply with the XSD standard. The xml file contains three types of tag structures, as shown in the table below:

S.No.	RPM Price Event Name	ORPOS Equivalent Name / Tag Structure Name
1	Price Change/Clearance	Price Change
2	Simple Promotion	Price Promotion
3	Threshold/Buy Get Promotion	Discount Rule

Assumptions and Scheduling Notes

RPMtoORPOSPublishExport should be executed as follows:

- After every successful run of RPMtoORPOSPublishBatch.
- Before PurgePayloadsBatch is run.

Primary Tables Involved

- RPM_ORPOS_PRICE_CHANGE_PUBLISH
- RPM_ORPOS_SIMPLE_PROMOTION_PUBLISH
- RPM_ORPOS_COMPLEX_PROMOTION_PUBLISH

Threading

RPMtoORPOSPublishExport is threaded. The LUW is a single store.

Configuration

The following property must be set in rpm.properties on the application server when using this batch program to publish all price events:

```
delete_staged_rib_payloads=false
```

statusPageCommandLineApplication Batch Design

The status page batch program (statusPageCommandLineApplication.sh) performs data checks to verify that some of the assumptions 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.frecon 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.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 PRMPRCCHG.PRMPRCCHGDEL is ON
**********The above exception indicates that we have passed
**************************
****************************************
Starting Report
RpmJmsServerCheck Passed

Done.
```

TaskPurgeBatch Batch Design

Overview

The TaskPurgeBatch program purges entries related to location move, conflict check, bulk conflict check and worksheet conflict check tables. In short, it purges all RPM_*TASK tables.

Usage

The following command runs the TaskPurgeBatch job:

```
taskPurgeBatch.sh <username> <password> [<purgeDays>] [Y/N]
```

Where the first argument is the user id. The second argument is the password, and the third argument is number of purge days (positive integer). The last argument is the complete status indicator (Y or N).

Detail

The TaskPurgeBatch purges the entries from RPM_*TASK tables based on the entered purge days and the status indicator. The purge days value is taken with respect to the VDATE in PERIOD table. All the entries before VDATE – purgeDays will be purged. The status indicator also is considered while the values are being purged. If the status indicator is Y, then only tasks with completed status will be purged. Otherwise, status is not considered while purging.

Assumptions and Scheduling Notes

TaskPurgeBatch can be run ad hoc.

Primary Tables Involved

- RPM_LOCATION_MOVE_TASK
- RPM_CONFLICT_CHECK_TASK
- RPM_BULK_CC_TASK
- RPM_WORKSHEET_CC_TASK
- RPM_TASK

Threading

TaskPurgeBatch is not threaded.

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. To qualify, the following criteria must be met:

- 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. To be processed, the following criteria must be met:

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

Each worksheet detail that meets the criteria above 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. Area differential logic also is executed.

If dynamic area differentials are being used in the system, any secondary area worksheet detail records that exist also are 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

WorksheetAutoApproveBatch is threaded but does not use the RPM_BATCH_CONTROL table. See the documentation for Bulk Conflict Check for thread configuration details.

ZoneFutureRetailPurgeBatch Batch Design

Overview

The ZoneFutureRetailPurgeBatch program deletes old error message from the price change auto approve batch program.

Usage

The following command runs the PriceChangeAutoApproveResultsPurgeBatch job:

```
ZoneFutureRetailPurgeBatch userid password
```

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

Detail

The ZoneFutureRetailPurgeBatch program deletes all zone/item price change actions that have an ACTION_DATE value prior to VDATE and have been superseded by at least one other change action for which ACTION_DATE also is prior to VDATE.

As a result, for each zone/item with a price change history, the most recent such action prior to VDATE remains after the purge is complete.

Assumptions and Scheduling Notes

ZoneFutureRetailPurgeBatch can be run ad hoc.

Primary Tables Involved

RPM_ZONE_FUTURE_RETAIL

Threading

ZoneFutureRetailPurgeBatch is not threaded.

RETL Program Overview for RPM Extractions

To facilitate the extraction of data from RPM (that could be loaded eventually into a data warehouse for reporting purposes, for example), RPM works in conjunction with the Oracle Retail Extract Transform and Load (RETL) framework. This architecture optimizes a high performance data processing tool that can let database batch processes take advantage of parallel processing capabilities. The RETL framework runs and parses through the valid operators composed in XML scripts.

Oracle Retail's streamlined RETL code provides for less data storage, easier implementation, and reduced maintenance requirements through decreased code volume and complexity. The RETL scripts are Korn shell scripts that are executable from a UNIX prompt. A typical run and debugging situation is provided later in this chapter.

These extractions initially were designed for Oracle Retail Data Warehouse (RDW), but they can be used for some other application in a retail enterprise.

For more information about the RETL tool, see the latest RETL Programmer Guide.

Architectural Design

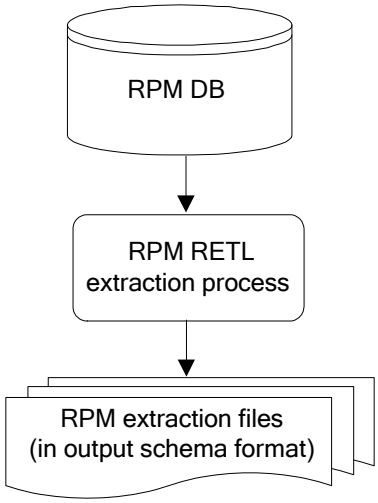
The diagram below illustrates the extraction processing architecture for RPM. Instead of managing the change captures as they occur in the source system during the day, the process extracts the current data from the source system. The extracted data is output to flat files. These flat files are then available for use by a product such as Oracle Retail Data Warehouse (RDW).

The target system (RDW, for example) transforms and loads the necessary data so that it can be used for further processing in the environment.

RPM modules use the same libraries, resource files and configuration files that RMS uses. All of these libraries, resource files and configure files are packed with RMS. An RPM retailer must install RMS before kicking off any RPM RETL scripts.

RPM Extraction Architecture

The architecture relies upon the use of well-defined flows specific to the RPM database. The resulting output comprises data files written in a well-defined schema file format. This extraction includes no destination-specific code.



RETL extraction processing for RPM

Configuration

RETL

RETL verion 12.0 (or later) is required to run RPM RETL and must be installed before RPM RETL can be configured and run. The verfify_retl script (included as part of the RETL installation) is run to ensure RETL is working properly before proceeding.

RETL User and Permissions

RPM ETL is installed and run as the RETL user. The permissions are set up per the RETL Programmer Guide. RPM ETL reads data and also creates, deletes and updates tables. If permissions are not set up properly, extractions fail.

Environment Variables

See the RETL Programmer Guide for RETL environment variables that must be set up for your version of RETL. The base directory for RPM must be set to MMHOME, the top level directory selected during the installation process. In .kshrc, add a line such as the following:

```
export MMHOME=<base directory for RMS ETL>\dwi12.0\dev
```

> **Note:** Because RPM modules share the same libraries and configure files with RMS, the MMHOME is the same as the one defined in RMS.

dwi_config.env Settings

Review the environmental parameters in the dwi_config.env file before executing batch modules. There are several variables to change, depending on local settings:

For example:

```
export DBNAME=int9i
export RPM_OWNER=steffej_RPM1101
export BA_OWNER=rmsint1101
export ORACLE_PORT="1524"
export ORACLE_HOST="mspdev38"
```

The environment variable PASSWORD can be set up in dwi_config.env--or in a different location specified by the retailer. In the example below, adding the line to the dwi_config.env causes the password, ”mypasswd”, to be used to log into the database:

```
export PASSWORD=mypasswd
```

Steps to Configure RETL

1. Log in to the Unix server with a Unix account that will run the RETL scripts.
2. Change directories to `$MMHOME/rfx/etc`.
3. Modify the dwi_config.env script:
   a. Change the DBNAME variable to the name of the RPM database.
   b. Change the RPM_OWNER variable to the username of the RPM schema owner.
   c. Change the BA_OWNER variable to the username of the RPME batch user.
   d. Change the ORACLE_HOST variable to the database server name.
   e. Change the ORACLE_PORT variable to the database port number
   f. Change the MAX_NUM_COLS variable to modify the maximum number of columns from which RETL selects records.

> **Note:** All RPM tables must be under the RMS database, which has the same BA_OWNER as RMS. A value must be assigned to RPM_OWNER in the RPM dwi_config.env file. The configuration file, dwi_config.env, as well as all other configure files are packed with RMS.

Program Features

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

Program Status Control Files

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

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

File Naming Conventions

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

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

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

For example, the program status control file for the prmevtex.ksh program would be named as follows for the VDATE of March 21, 2004:

```
$MMHOME/error/prmevtex.status.20040321
```

Restart and Recovery

Because RETL processes all records as a set (rather than one record at a time) the method for restart and recovery must be different from the method that is used for Pro*C. The restart and recovery process serves two purposes:

- It prevents the loss of data due to program or database failure.
- It increases performance when restarting after a program or database failure by limiting the amount of reprocessing required.

The RPME extract modules extract from a source transaction database and write to a text file.

Most modules use a single RETL flow and do not require the use of restart and recovery. If the extraction process fails for any reason, the problem can be fixed, and the entire process can be run from the beginning without the loss of data.

There is no restart/recovery logic in any RPM RETL extraction module.

Message Logging

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

Daily Log File

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

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

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

For example, the location and the name of the log file for the business virtual date (VDATE) of March 21, 2004 would be the following:

```
$MMHOME/log/20040321.log
```

Format

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

```
prmevtex 15:47:14: Program started...
prmevtex 15:47:18: Program completed successfully
```

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

Program Error File

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

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

The naming convention for the error file defaults to the following dot-separated file name:

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

For example, all errors and detail log information for the prmevtex.ksh program would be placed in the following file for the batch run of March 21, 2004:

```
$MMHOME/error/prmevtex.20040321
```

Schema Files

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

Resource Files

RPME Kornshell programs use resource files so that the same RETL programs can run in various language environments. For each language, there is one resource file.

Resource files contain hard-coded strings that are used by extract programs. The name and directory of the resource file is set in the configuration file (dwi_config.env). The default directory is ${MMHOME}/rfx/include.

The naming convention for the resource file follows the two-letter SIM code standard abbreviation for languages (for example, en for English, fr for French, ja for Japanese, es for Spanish, de for German, and so on).

Note: Resource files are only packed with RMS.

Typical Run and Debugging Situations

The following examples illustrate typical run and debugging situations for types of programs. The log, error, and other file names referenced below assume that the module is run on the business virtual date of March 9, 2004. See the previously-described naming conventions for the location of each file.

For example:

To run prmevtex.ksh:

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

```
%prmevtex.ksh
```

If the module runs successfully, the following results:

- Log file
  Today's log file, 20040309.log, contains the messages "Program started …" and "Program completed successfully" for prmevtex.ksh.
- Data
  The prmevtdm.txt file exists in the $MMHOME/data directory and contains the extracted records.
- Error file
  The error file, prmevtex.20040309, contains the standard RETL flow (ending with "All threads complete" and "Flow ran successfully") and no additional error messages.
- Program status control
  The program status control file, prmevtex .status.20040309, does not exist.

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

- Log file
  The current log file, 20040309.log, does not contain the "Program completed successfully" message for prmevtex.ksh.
- Data
  The prmevtdm.txt file may exist in the data directory but may not contain all the extracted records.
- Error file
  The error file,prmevtex.txt.20040309, may contain an error message.
- Program status control
  The program status control file, prmevtex.status.20040309, exists.

To re-run a module from the beginning, complete the following steps:

1. Determine and fix the problem causing the error.
2. Remove the status control file for the program.
3. Change directories to $MMHOME/rfx/src. At a Unix prompt, enter:

```
%prmevtex.ksh
```

> **Note:** To understand how to engage in the restart and recovery process, see the section, "Restart and recovery" earlier in this chapter.

RETL Extractions Program List

This section describes the RETL extraction RPM programs.

> **Note:** See the Oracle Retail Merchandising Batch Schedule for information on the batch schedule and program flow diagrams.

Program	Functional Area	Source Table or File	Schema File	Target File or Table
prmdtlex.ksh	Promotion	RPM_PROMO_COMP, RPM_PROMO_COMP_BUY_GET, RPM_PROMO_COMP_DETAIL, RPM_PROMO, RPM_PROMO_COMP_THRESHOLD, RPM_PROMO_COMP_SIMPLE	prmdtldm.schema	prmdtldm.txt
prmevtex.ksh	Promotion	RPM_PROMO_EVENT	prmevtdm.schema	prmevtdm.txt
prmhdrex.ksh	Promotion	RPM_PROMO	prmhdrdm.schema	prmhdrdm.txt

Application Programming Interface (API) Flat File Specifications

This section contains APIs that describe the file format specifications for all text files.

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

API Format

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

File Layout

- Field Name: Provides the name of the field in the text file.
- Description: Provides a brief explanation of the information held in the field.
- Data Type/Bytes: Includes both data type and maximum column length. Data type identifies one of three valid data types: character, number, or date. The Bytes section identifies the maximum bytes available for a field. A field may not exceed the maximum number of bytes (note that ASCII characters usually have a ratio of 1 byte = 1 character).
  - Character fields can hold letters (such as A, B, c, and d), numbers (such as 1,2, and 3), and special characters (such as $,#, and &).
  - Number fields can hold only numbers (such as 1,2, and 3).
  - Date fields hold numeric values representing a year, month, day combination. Unless otherwise specified, date format is "YYYYMMDD" (for example, 20041014 for 14 October 2004).
- Any required formatting for a field is conveyed in the Bytes section. For example, Number(18,4) refers to number precision and scale. The first value is the precision and always matches the maximum number of digits for that field; the second value is the scale and specifies, of the total digits in the field, how many digits exist to the right of the decimal point. For example, the number –12345678901234.1234 would take up twenty ASCII characters in the flat file; however, the overall precision of the number is still (18,4).
- Field Order: Identifies the order of the field in the schema file.
- Required Field: Identifies whether the field must contain a non-null value. Valid values are Y and N. Yes signifies the field must contain a non-null value and may not hold a null value. No signifies that the field may hold a null value.

General Business Rules and Standards Common to all APIs

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

  If there are no records for the day, an empty flat file must still be provided.
- Fact data loaded to RDW can be either new or updated facts. Unlike dimension snapshots, fact flat files contain only new/updated facts exported from the source system once per day (or week, in some cases). Refer to each API specific business rules section for more details.

  If there are no new or changed records for the day, an empty flat file must still be provided.
- Amounts are stored in both primary and local currencies for most fact tables. If the source system uses multi-currency, then the primary currency column holds the primary currency amount, and the local currency column holds the local currency amount. If the location happens to use the primary currency, then both primary and local amounts hold the primary currency amount. If the source system does not use multi-currency, then only the primary currency fields are populated--and the local fields hold NULL values.
- Values entered into the text files are the exact values processed and loaded into the datamart tables. Therefore, the values with leading and/or trailing zeros, characters, or nulls are processed as such. RDW does not strip any of these leading or trailing values, unless otherwise noted in the individual API business rules section.
- It is assumed that indicator columns hold one of two values, Y or N.

- Delimiters:

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

  - Dimension Flat File Delimiter Standards (as defined by RDW): Within dimension text files, each field must be separated by a pipe (|) character, for example a record from prddivdm.txt may look like the following:

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

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

    ```
    WIS;20010311;1.73527820592648544918
    ```

    See the latest RETL Programmer Guide for additional information.
- Each record in the text file must be separated by an end-of-line carriage return. For example, three records, each holding four values, should be entered like this:

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

  and not as a continuous string of data, like this:

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

prmdtldm.txt

Business rules:

- This interface file cannot contain duplicate records for an event_idnt, head_idnt, prmtn_dtl_idnt combination.
- This interface file contains the complete snapshot of active information.
- If a dimension identifier is required but is not available, a value of -1 is needed.
- This interface file follows the dimension flat file interface layout standard.
- event_idnt will be -2 for promotion without event.

Name	Description	Data Type/Bytes	Field order	Required field
PRMTN_DTL_IDNT	The unique identifier of a promotion detail.	VARCHAR2(10)	1	Yes
HEAD_IDNT	The unique identifier of a promotion head.	VARCHAR2(10)	2	Yes
EVENT_IDNT	The unique identifier of a promotion event.	VARCHAR2(10)	3	Yes

Name	Description	Data Type/Bytes	Field order	Required field
PRMTN_TRIG_TYPE_IDNT	The unique identifier of the promotion trigger type. Valid values can be 'offer code', 'media code', and so on.	NUMBER(10)	4	Yes
PRMTN_SRC_CDE	The unique identifier of a promotion source. Valid values can be 'DTC', 'RPM' or any other promotion source chosen by client.	VARCHAR2(6)	5	Yes
PRMTN_SVC_TYPE_IDNT	The unique identifier of a promotion service type.	VARCHAR2(10)	6	Yes
PRMTN_FMT_IDNT	The unique identifier of a promotion format.	VARCHAR2(10)	7	Yes
BEG_DT	The date the promotion begins.	DATE	8	Yes
PRMTN_DTL_DESC	Description for the promotion detail identifier.	VARCHAR2(160)	9	No
PRMTN_SVC_TYPE_DESC	Description for the promotion service type.	VARCHAR2(120)	10	No
PRMTN_FMT_DESC	Description for the promotion format.	VARCHAR2(120)	11	No
END_DT	The date the promotion ends.	DATE	12	No

prmevtdm.txt

Business rules:

- This interface file contains promotion events and related attributes. Events are time periods used to group promotions for analysis.
- This interface file cannot contain duplicate records for an event_idnt.
- This interface file follows the dimension flat file interface layout standard.
- This interface file contains the complete snapshot of active information.

Name	Description	Data Type/Bytes	Field order	Required field
EVENT_IDNT	The unique identifier of a promotion event.	VARCHAR2(10)	1	Yes
EVENT_DESC	Description for the promotion event.	VARCHAR2(1000)	2	No
THEME_DESC	Description for the promotion theme.	VARCHAR2(120)	3	No

prmhdrdm.txt

Business rules:

- This interface file contains promotion headers and their attributes. Headers define a promotion and its start/end dates.
- This interface file cannot contain duplicate records for a head_idnt.
- All promotion head_idnt records require a beginning date, even if they are dummy values such as 4444-04-04.
- This interface file follows the dimension flat file interface layout standard.
- This interface file contains the complete snapshot of active information.
- event_idnt is -2 for promotion without event.

Name	Description	Data Type/Bytes	Field order	Required field
HEAD_IDNT	The unique identifier of a promotion head.	VARCHAR2(10)	1	Yes
EVENT_IDNT	The unique identifier of a promotion event.	VARCHAR2(10)	2	Yes
HEAD_NAME	Name for the promotion head.	VARCHAR2(120)	3	No
HEAD_DESC	Description for the promotion head.	VARCHAR2(255)	4	No
BEG_DT	The date the promotion begins.	DATE	5	Yes
END_DT	The date the promotion ends.	DATE	6	No