Oracle® Retail Price Management

Operations Guide

Release 13.2.4.0.1

E35694-01

June 2012

ORACLE®

Oracle Retail Price Management Operations Guide, Release 13.2.4.0.1

Primary Author: Kris Lange

Value-Added Reseller (VAR) Language

Oracle Retail VAR Applications

The following restrictions and provisions only apply to the programs referred to in this section and licensed to you. You acknowledge that the programs may contain third party software (VAR applications) licensed to Oracle. Depending upon your product and its version number, the VAR applications may include:

(i) 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.

(ii) the **Wavelink** component developed and licensed by Wavelink Corporation (Wavelink) of Kirkland, Washington, to Oracle and imbedded in Oracle Retail Mobile Store Inventory Management.

(iii) 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.

(iv) 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.

You acknowledge and confirm that Oracle grants you use of only the object code of the VAR Applications. Oracle will not deliver source code to the VAR Applications to you. Notwithstanding any other term or condition of the agreement and this ordering document, you shall not cause or permit alteration of any VAR Applications. For purposes of this section, "alteration" refers to all alterations, translations, upgrades, enhancements, customizations or modifications of all or any portion of the VAR Applications including all reconfigurations, reassembly or reverse assembly, re-engineering or reverse engineering and recompilations or reverse compilations of the VAR Applications or any derivatives of the VAR Applications. You acknowledge that it shall be a breach of the agreement to utilize the relationship, and/or confidential information of the VAR Applications for purposes of competitive discovery.

The VAR Applications contain trade secrets of Oracle and Oracle's licensors and Customer shall not attempt, cause, or permit the alteration, decompilation, reverse engineering, disassembly or other reduction of the VAR Applications to a human perceivable form. Oracle reserves the right to replace, with functional equivalent software, any of the VAR Applications in future releases of the applicable program.

Contents

3 Technical Architecture

4 Conflict Checking

5 Integration Methods and Communication Flow

6 Functional Design

7 Java Batch Processes

Send Us Your Comments

Oracle Retail Price Management Operations Guide, Release 13.2.4.0.1

Oracle welcomes customers' comments and suggestions on the quality and usefulness of this document.

Your feedback is important, and helps us to best meet your needs as a user of our products. For example:

- Are the implementation steps correct and complete?
- Did you understand the context of the procedures?
- Did you find any errors in the information?
- Does the structure of the information help you with your tasks?
- Do you need different information or graphics? If so, where, and in what format?
- Are the examples correct? Do you need more examples?

If you find any errors or have any other suggestions for improvement, then please tell us your name, the name of the company who has licensed our products, the title and part number of the documentation and the chapter, section, and page number (if available).

> **Note:** Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the new Applications Release Online Documentation CD available on My Oracle Support and www.oracle.com. It contains the most current Documentation Library plus all documents revised or released recently.

Send your comments to us using the electronic mail address: retail-doc_us@oracle.com

Please give your name, address, electronic mail address, and telephone number (optional).

If you need assistance with Oracle software, then please contact your support representative or Oracle Support Services.

If you require training or instruction in using Oracle software, then please contact your Oracle local office and inquire about our Oracle University offerings. A list of Oracle offices is available on our Web site at www.oracle.com.

Preface

This Operations Guide provides critical information about the processing and operating details of Oracle Retail Price Management, including the following:

- System configuration settings
- Technical architecture
- Functional integration dataflow across the enterprise
- Batch processing

Audience

This guide is for:

- Systems administration and operations personnel.
- Systems analysts.
- Integrators and implementers.
- Business analysts who need information about Product processes and interfaces.

Related Documents

For more information, see the following documents in the Oracle Retail Price Management Release 13.2.4.0.1 documentation set:

- *Oracle Retail Price Management Release Notes*
- *Oracle Retail Price Management Installation Guide*
- *Oracle Retail Price Management User Guide*
- *Oracle Retail Price Management Online Help*
- *Oracle Retail Price Management Operations Guide*
- *Oracle Retail Price Management Data Model*
- *Oracle Retail Merchandising Batch Schedule*
- *Oracle Retail Merchandising Implementation Guide*
- *Oracle Retail POS Suite 13.4.1/MOM 13.2.4 Implementation Guide*

Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:

https://support.oracle.com

When contacting Customer Support, please provide the following:

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

Review Patch Documentation

When you install the application for the first time, you install either a base release (for example, 13.1) or a later patch release (for example, 13.1.2). If you are installing the base release, additional patch, and bundled hot fix releases, read the documentation for all releases that have occurred since the base release before you begin installation. Documentation for patch and bundled hot fix releases can contain critical information related to the base release, as well as information about code changes since the base release.

Oracle Retail Documentation on the Oracle Technology Network

Documentation is packaged with each Oracle Retail product release. Oracle Retail product documentation is also available on the following Web site:

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

(Data Model documents are not available through Oracle Technology Network. These documents are packaged with released code, or you can obtain them through My Oracle Support.)

Documentation should be available on this Web site within a month after a product release.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1
Introduction

RPM is a pricing and promotions execution system. RPM's functionality includes the definition, maintenance, and review of price changes, clearances and promotions. The system's capabilities range from simple item price changes at a single location to multi-buy 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 RPM's 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.

RPM pricing strategies allow the user to define clearance and regular item retails based on rules-based logic. Price changes are proposed in a pricing worksheet, and the user reviews and accepts them. The pricing worksheet gives the retailer access to future and price events at the item/zone level, which can be helpful in making pricing decisions. Depending on the items affected, pricing strategies can be defined at the department, class or subclass level. Types of strategies include margin based, competitor driven and proposed (as clearance markdowns).

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, but is meant to provide descriptive overviews of the key system parameters.

Supported Environments

See the *Oracle Retail Price Management 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 can bring the system to a halt. For example, the connection to the server is lost.

- Business exceptions

  This exception indicates that a business rule has been violated. Most exceptions that arise in the system are business exceptions. For example, a user tries to approve a price change that causes a negative retail.

Configuration Files

Key system configuration parameters are described in this section. Many parameters have been omitted from this section that retailers should not have to change. When retailers install RPM into an environment, they must update these values to their specific settings.

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

RPM application installer configures all data source settings for the RPM application. To change the data source settings for the RPM application after it has been deployed, complete the following steps:

1. Log into the WebLogic Server 11g Administration Console. Navigate to JDBC -> Data Sources.
2. The data sources listed are RPMNonXADataSource and RPMXADataSource.
3. Click **RPMNonXADataSource.**
4. Go to the Connection Pool tab.
5. Make any necessary changes to the JDBC URL and user name. Click **Save**.
6. Repeat the same steps above for RPMXADataSource.
7. If you are changing the schema owner, you must also make this change in the rpm.properties file in the deployment. Log in to the UNIX server and change directories to:

   <ORACLE_HOME>//user_projects/domains/soa_domain/servers/rpm-server/tmp/_WL_user/<rpm_app_name>/<encrypted_folder>/conf

   Modify the rpm.properties file with the new schema_owner value. This value must be in all capital letters. Save the file.
8. Restart the managed server 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 following value. For more information about the RIB, see Chapter 5, "Integration Methods and Communication Flow," and RIB documentation.

- rib.user=valid user for the system

security.properties

There must be a security.properties file located in conf/retek. This properties file is used to configure the embedded Oracle Retail Security Manager (RSM). See "Application Security" in Chapter 6, "Functional Design."

The file is populated with the following values.

Wallet Settings

The following properties pertain to wallet setup.

- csm-wallet.path = <ORACLE_HOME>/user_projects/domains/soa_domain/retail/<application_name>/config
- csm-wallet.partition = rpm13

For LDAP Authentication

These values are used for the configuration of the authentication process as it is run through LDAP. Once an LDAP schema is established, a retailer enters applicable LDAP properties to point to that schema.

ldap.initialcontextfactory

This internal Java-specific setting should not change from its initial value.

For example:

```
ldap.initialcontextfactory=com.sun.jndi.ldap.LdapCtxFactory
```

ldap.authenticationprovider.url

This value represents the authentication provider's URL. In a production environment, this setting would contain the retailer's address for its directory server.

For example:

```
ldap.authenticationprovider.url=ldap://64.238.67.60:379/
```

ldap.user.basedn

The values in this entry must correspond to entries in the LDAP server. DN stands for distinguished name. The top level of the LDAP directory tree is the base, referred to as the base DN. This value represents the user base DN property.

For example:

```
lldap.user.basedn=cn=users,dc=us,dc=oracle,dc=com
```

ldap.authenticationmode

This value represents the authentication mode property. LDAP uses various ways to authenticate against a directory server, and the method of authentication can be set up. For almost all environments integrated with RSM, the value should be simple.

For example:

```
ldap.authenticationmode=simple
```

ldap.securityprotocol

Note: This setting is currently not used by RSM.

This value represents RSM's encryption protocol. SSL stands for secure socket layer (SSL). SSL is a protocol developed for private transmissions. SSL works by using a private key to encrypt data that's transferred over the SSL connection.

For User Search

These settings provide the behind the scenes login information for the system to connect to the directory server. For example, when an RSM user wishes to search on the directory server for a user, the RSM system must have a user name and password to log in to the directory server to enable the search to occur. The password for the LDAP server is maintained in the wallet. RSM uses the useralias to obtain the password from the wallet. The filter property value represents the directory server-specific way of filtering user information by attribute (when the directory server is finding users and then limiting the results). Because various directory servers use different attributes to represent a user name, this value must be updated if the retailer were to change directory servers. For example:

```
ldldap.usersearch.user=cn=RPM.ADMIN,cn=users,dc=us,dc=oracle,dc=com
ldap.usersearch.useralias=RPM.ADMIN
ldap.user.filter=(&(objectCategory=person)(objectClass=user) %v)
```

For Audit Logging

For audit logging, the audit.logger setting is used. This setting allows you to direct security audit information to a specific Log4J logger. This value must match a logger/appender in the RSM server log4j.xml file. If a match does not occur, the root logger and appender are used.

For example:

```
audit.logger=Security.Audit.Logger
```

Single Sign-On with Oracle Technology

For single sign-on, the enable.oracle.sso setting is used. The value for this setting should always be false. For example:

```
enable.oracle.sso=false
```

LoginModule Configuraton Information

This setting configures RSM to point to the applicable user repository (such as a directory server or xml file) for authentication. The login modules defined in the application server can be found in rpm_jaas.config. See the Oracle Retail Price Management Installation Guide for detailed information pertaining to login module configuration. For example:

```
loginmodule=Oracle.LoginModule
```

The following example illustrates the entry in rpm_jaas.config for an authentication against an LDAP compliant directory server

```
Oracle.LoginModule{
com.retek.rsm.domain.security.dao.LdapLoginModule required debug=true
```

The following example illustrates the entry in rpm_jaas.config for an authentication against the RSM users XML file:

```
Oracle.LoginModule{
com.retek.rsm.domain.security.dao.XMLLoginModule required debug=true
```

Note: This setting must correspond with the user DAO implementation setting in the dao_rpm.xml.

If the XMLLoginModule is used, users must be added to the file, users_rsm.xml.

For Mapping LDAP to Directory Schema

The following table contains directory server-specific attributes concerning user information.

Various directory servers use different attributes to represent user information. If a retailer were to change directory servers, these values must be configured to reflect the new directory server.

Element	Definition
ldap.firstname.attrname	LDAP first name attribute name property
ldap.lastname.attrname	LDAP last name attribute name property
ldap.username.attrname	LDAP user name attribute name property

User Signature Information

To facilitate single sign on functionality, a user signature may be passed among a retailer's RSM-integrated applications. The following steps describe how the user signature is created and could be used:

1. When a user first logs in to an Oracle Retail application secured by RSM, it sends RSM the user and password data required for authentication.
2. RSM calls the retailer's LDAP compliant directory service to authenticate the user name and password data. Once a user is authenticated, RSM creates an encrypted user signature, which is returned to the calling Oracle Retail application.
3. When the user launches RPM, for example, the user signature is passed to RPM. The RPM application accepts the user signature and calls RSM to determine whether the signature is valid. If the validation step is successful, the user accesses RPM without having to go through the RPM login screen.

user.signature.cipher.algorithm

RSM uses an algorithm to generate a user signature. A retailer may change this algorithm and configure this property value to reflect the different algorithm being used.

For example:

```
user.signature.cipher.algorithm=HmacSHA1
```

user.signature.secretkey

To generate user signatures, the algorithm needs a secret key. Oracle Retail recommends that the retailer updates this value on a regular basis. A retailer can change this secret key if a compromise in security has occurred.

For example:

```
user.signature.secretkey=gjgh6382nEDmxMLc3DSkhYP0ah347495
```

user.signature.salt

The system uses the salt value to avoid dictionary attacks. Salt adds characters to what is being created (in this case, a user signature). Because of the salt value, for example, the encrypted value might have 100 digits rather than 10 digits. Breaking the encryption thus becomes more difficult.

For example:

```
user.signature.salt=¥!asdfghlll©ñ¤?#¥³1966
```

User Authentication Information

user.max.allowable.authentication.failures

This value represents the maximum number of times that a user can fail authentication before the user's account is locked.

For example:

```
user.max.allowable.authentication.failures=5
```

user.max.time.lock.useraccount

This value represents the maximum number of hours a user's account remains locked. If the account is locked and over the maximum time value, the next time that user logs onto the system, the lock releases.

For example:

```
user.max.time.lock.useraccount=30
```

dao_rpm.xml

There must be a dao_rpm.xml file located in conf/retek. This configuration file is used to configure the mapping between DAO interfaces and their implementation. Generally there is no need to make changes to this file except if there is a need to configure the user repository that is used by RSM for user searches. The default value is to use an LDAP compliant directory server as the user repository. Besides LDAP, XML file based searches are also supported. To switch between LDAP and XML, comment or uncomment following the impl package tags.

> **Note:** This setting should correspond with the LoginModule configuration information found in the rpm_jaas.config file.

For example:

```
<dao-config>
<customizations>
<!-- There can only be one impl per interface.  Use the XMLImpl to xml file(s) as
the user repository.  -->
<interface package="com.retek.rsm.domain.security.dao.user">
<impl  package="com.retek.rsm.domain.security.dao.impl.user" prefix=""
suffix="LDAPImpl"/>
<!--
<impl  package="com.retek.rsm.domain.security.dao.impl.user" prefix=""
suffix="XMLImpl"/>
```

```
-->
    </interface>
</customizations>
</dao-config>
```

users_rsm.xml

There is a users_rsm.xml file located in conf/retek. This XML configuration is used for authentication and user searching, this file is used as the repository for the users. This file must contain the user names, first names, last names of all valid users. The corresponding password for the user is sourced from the wallet file. This file is located in the conf/retek directory within the RPM ear file.

> **Note:** If LDAP is used for authentication and user searching, this file is ignored.

For example:

```
<<users>
<user username="Valid.User" firstname="Valid" lastname="User" />
<user username="JoeUser" firstname="Joe" lastname="User" />
</users>
```

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

RPM's service factory configuration file, services_rpm.xml, specifies the mapping between RPM's 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 5, "Integration Methods and Communication Flow."

The following example illustrates how the businesslogic flavorset is indicated in the service factory configuration file.

```
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>
```

Logging

Logging within RPM takes place in a number of ways, as the following describes.

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 server's flat text log file. This logging mechanism reveals errors and other significant events that occur during the system's 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>/user_projects/domains/soa_domain/servers/<rpm_server>/tmp/_WL_user/<rpm_app_name>/<encrypted_folder>/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 configuration, the output is written to a file in this format RPM.log under <ORACLE_HOME>/user_projects/domains/sso_domain/servers/RPM132_MS/logs. You can also configure to write logs to a different location and to roll them according to file size. For instructions related to this procedure, see the WebLogic Configuration and Administration Guide.

Hibernate Logging

Hibernate's 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, please follow these steps:

1. Log in to the WebLogic Server 11g Administration Console.
2. Under Services, click **JTA.**
3. Click the **Configuration** tab.
4. Under JTA, set the Timeout Seconds (for example, 600 seconds).

RPMTaskMDB

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

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

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.

RPMChunkMDB

RPMChunkMDB is a message driven bean used for RPM's chunking functionality. Chunking refers to the breaking up of large price events (such as department level promotions) into smaller subsets. When chunking is required, the system publishes a message to the RPM chunk queue. Messages received by the chunk queue are processed through RPMChunkMDB.

Tables Used by RPMChunkMDB

RPM_CHUNK_CC_TASK and RPM_TASK

Multi-threading of MDB

Multi-threading for RPMTaskMDB and RPMChunkMDB is configured by the weblogicdeployment descriptor, weblogic-ejb-jar.xml.

To configure the number of threads, complete these steps:

1. Update the MDB descriptor in weblogic-ejb-jar.xml to add the following

```
<pool>
              <max-beans-in-free-pool>10</max-beans-in-free-pool>
       </pool>
```

   For example, for RPMTaskMDB it will be:

```
<message-driven-descriptor>
      <pool>
              <max-beans-in-free-pool>10</max-beans-in-free-pool>
       </pool>
         <destination-jndi-name>jms/taskQueue</destination-jndi-name>
...
</message-driven-descriptor>
```

2. Note that <pool> tag should be the first tag after <message-driven-descriptor>.

Configuring RPM without the RIB

RPM integrates with RMS through the RIB. However, depending on the type of data being passed, the RIB is not always required. A single system option, RPM_IND, determines first whether integration between RPM and RMS is required by the retailer and, second, whether the RIB is required for each data flow. Setting RPM_IND to Y indicates only that an interface with RPM and RMS is necessary; the system then decides whether the RIB is needed. See the *Oracle Retail Merchandising System Installation Guide* for additional information on setting the value of the system_options table.

1. Log in to the UNIX server as the user who has write access under ORACLE_HOME.
2. Change directories to <ORACLE_HOME>/user_projects/domains/sso_domain/config.
3. Make a backup of oconfig.xml and edit the file. Locate the managed server running the RPM application.
4. Under the managed server, under <server-start>-> <arguments>, add the following value:

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

   For example:

   ```
   <<server-start>
         <class-path>…</class-path>
         <arguments>… -Dretek.no.rib=false</arguments>
       </server-start>
   ```

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 need to be enabled so that data is 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:

- Delete_staged_rib_payloads=true | false (default is true): configured in rpm.properties
- retek.no.rib=true | false (default is false): configured through JVM system property
- retek.no.rib.publish=true | false (default is false): configured through JVM system property

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

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>	Yes	No	No
FALSE	FALSE	FALSE	Yes	Yes	Yes
FALSE	FALSE	TRUE	Yes	No	Yes
FALSE	TRUE	<ANY>	Yes	No	Yes

Do the following to set the options for the different results:

- 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 need to be turned off from the integrated batch schedule:

- NewItemLocBatch
- ItemLocDeleteBatch

Disabling RIB Publishing in RPM

The following steps 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 5, "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>/user_projects/domains/sso_domain/config.
3. Make a backup of config.xml and then edit the file. Locate managed server running the RPM application.
4. Under the managed server, under <server-start>-> <arguments>, add the following value:

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

   For example:

   ```
   <<server-start>
         <class-path>…</class-path>
         <arguments>… -Dretek.no.rib=true</arguments>
       </server-start>
   ```

5. Save config.xml.
6. Restart the managed server instance running the RPM application.

Internationalization

Internationalization is the process of creating software that can be translated more easily. Changes to the code are not specific to any particular market. RPM has been internationalized to support multiple languages.

This section describes configuration settings and features of the software 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

The following components are not translated:

- Documentation (online help, release notes, installation guide, user guide, operations guide)
- Batch programs and messages
- Log files
- Configuration tools
- Reports
- Demonstration data
- Training materials

The user interface for RPM has been translated from U.S. English into the following languages:

- Chinese (simplified)
- Chinese (traditional)
- Croatian
- Dutch
- French
- German
- Greek
- Hungarian
- Italian
- Japanese
- Korean
- Polish
- Portuguese (Brazilian)
- Russian
- Spanish

- Swedish
- Turkish

Set the Client Operating System to the Applicable Locale

For a client machine to use the translated interface, set the operating system to the appropriate locale. The following is the procedure for setting a Microsoft Windows XP OS to a particular language. For other operating systems, 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 is displayed.
2. Select the required language from the Standards and formats drop-down field.
3. Click **OK.**

Resources_xx.properties

The text in the following .properties file is translated so that the interface functions in local settings. When a country code other than the default is used, the retailer populates the following _xx.properties files, 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 the following shows, the properties files can be found in rpm_client_properties.jar and rpm_server_properties.jar.

- rpm_client_properties.jar
  - rpm13-ui/src/com/retek/rpm/gui/Resources.properties
  - rpm13-ui/src/com/retek/rpm/gui/worksheet/Worksheet_column_names.properties
- rpm_server_properties.jar
  - rpm13-server/conf/retek/messages.properties
  - rpm13-server/conf/retek/codes.properties
  - rpm13-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.

Adjusting the Default Font from Settings to Ensure Proper Display of Asian Characters

Within the RPM application, Asian characters do not display correctly when the font setting, Verdana, is used. This condition exists for clients who use mixed language sets, where the user interface is in a European language (for example, English) and data field values are in an Asian language.

To ensure proper display of Asian characters, the font setting must be set to Dialog. The font setting is found in the Resources.properties file (for English language defaults) and in the Resources._xx.properties file (for all other European language defaults).

Adjust the Default Font Settings

Adjusting the default font settings requires modification of the Resources.properties (or Resources_xx.properties) file in platform-resources.jar. The following skills and tools are necessary to complete the modification:

- Thorough knowledge of the RPM application
- Thorough knowledge of Java programming
- Java programming toolkit

The platform-resources.jar must be updated in both of the following locations :

- For Client: $ORACLE_HOME/user_projects/domains/sso_domain/servers/<managed_server>/tmp/_WL_user/<rpm_instance>/<encrypted_folder_name>/war/client/lib folder
- For Server: $ORACLE_HOME/user_projects/domains/sso_domain/servers/<managed_server>/tmp/_WL_user/<rpm_instance>/<encrypted_folder_name>/lib folder

> **Note:** To complete the following steps, you must have a Java executable in your PATH to use the jar and jarsigner commands. To confirm that it is there, enter the command, which java, and note whether a valid Java executable file returns.

The following are the steps required to modify the Resources.properties (or Resources_xx.properties) file in the platform-resources.jar:

1. Back up the original platform-resources.jar file (for example, $ cp platform-resources.jar platform-resources.jar.ORIG).
2. Make a temporary directory (for example, $mkdir jar_temp).
3. Move the original platform-resources.jar file into the temporary directory (for example, mv platform-resources.jar jar_temp/).
4. Using a command such as jar -xvf platform-resources.jar, extract the platform-resources.jar.
5. Modify the Resources.properties (or Resources_xx.properties) file as follows:

> **Note:** When changing the font to Dialog, clients may opt to turn off bolding within the user interface. Bolded field labels can sometimes be hard to read, because the characters become crowded on the form. Unbolded characters can be easier to read. Settings for both options are provided here.

- Option 1: Change the font to Dialog, without bolding.

  Modify the properties as follows:

  ```
  defaultFontName=dialog
  defaultBoldFontName=dialog
  defaultBoldFontStyle=0
  defaultSmallFontName=dialog
  defaultSmallBoldFontName=dialog
  defaultSmallBoldFontStyle=0
  defaultLargeBoldFontName=dialog
  defaultLargeBoldFontStyle=0
  menuAcceleratorFontName=dialog
  ```

- Option 2: Change the font to Dialog, with bolding.

  Modify the properties as follows:

  ```
  defaultFontName=dialog
  defaultBoldFontName=dialog
  defaultBoldFontStyle=1
  defaultSmallFontName=dialog
  defaultSmallBoldFontName=dialog
  defaultSmallBoldFontStyle=1
  defaultLargeBoldFontName=dialog
  defaultLargeBoldFontStyle=1
  menuAcceleratorFontName=dialog
  ```

6. Changing the font style from Verdana to Dialog may also require changing the font size. Update the following font size properties as needed:

   ```
   defaultFontSize
   defaultBoldFontSize
   defaultSmallFontSize
   defaultSmallBoldFontSize
   defaultLargeBoldFontSize
   menuAcceleratorFontSize
   ```

7. Unsign the platform-resources.jar file by removing the following ORACLE* files from the extracted jar file:
   - META-INF/ORACLE_C.SF
   - META-INF/ORACLE_C.RSA
8. Using a command such as jar -xvf platform-resources.jar, jar the remaining files in the extracted file.
9. Place the newly modified platform-resources.jar in the location of the original jar file.
10. Re-sign the client jar file. For information on how to sign the platform-resources.jar, see the section, "Sign the RPM Client Configuration Jar File," in the *Oracle Retail Price Management Installation Guide*.

    Note: All client-side jars containing Java code must be signed with the same signature.

11. Restart the application server.

Price Management Status Page

Because RPM is dependent upon a number of servers, and a number of Oracle Retail products are dependent on RPM, a status page helps the retailer determine quickly whether RPM and the servers upon which it depends are up and running correctly. The privileges to this page can be set in Oracle Retail Security Manager and these privileges are typically reserved for administrators. The status page application displays the answers to the following questions:

- Is the RPM/RMS_Database up and running?
- Is the RPM JMS Server up and running?
- Is RSM up and 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 following text represents a sample output. The example represents a case in which the questions above have all been answered in the affirmative.

The new command usage is as follows:

```
statusPageCommandLineApplication.sh userAlias [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 retailUser 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_
76208b17_Tie.executeCommand__com_retek_platform_service_ServiceCommand(_
EJSRemoteStatelessCommandExecutionService_76208b17_Tie.java(Compiled Code))
at com.retek.platform.service.impl._EJSRemoteStatelessCommandExecutionService_
76208b17_Tie._invoke(_EJSRemoteStatelessCommandExecutionService_76208b17_
Tie.java(Compiled 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. is ON
************************The above exception indicates that we have passed
**************************
****************************************
Starting Report
com.retek.rpm.statuspage.RpmRibMessageCheck Passed PRMPRCCHG. is ON
************************The above exception indicates that we have passed
**************************
****************************************
Starting Report
com.retek.rpm.statuspage.RpmRibMessageCheck Passed PRMPRCCHG. 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 for your reference at the end of this chapter.

Overview of RPM Architecture

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

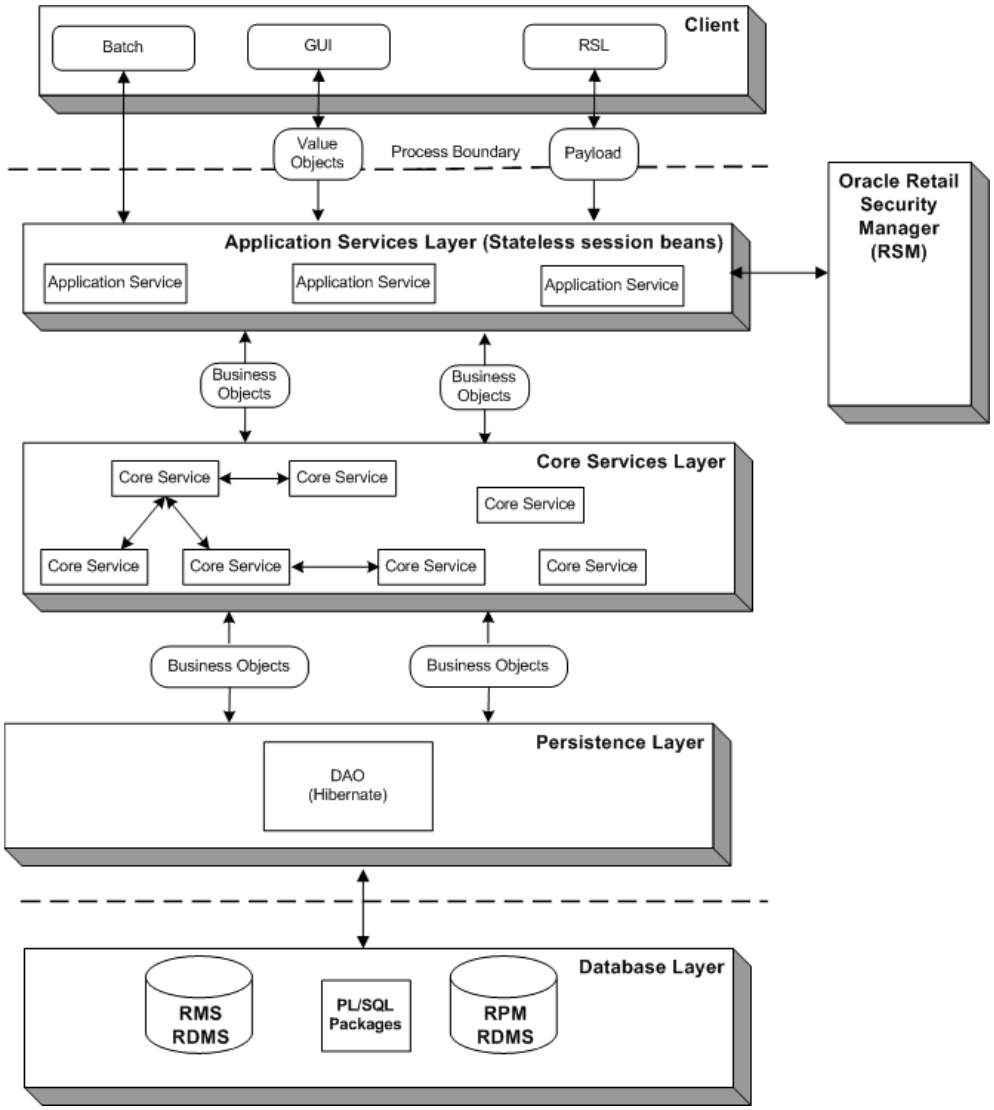
Conceptually, RPM's J2EE architecture is built upon 4-layers and implements what is defined as a service-oriented architecture. Such an architecture is essentially a collection of services that pass data, perform business processing, coordinate system activities, and render data into abstract objects. Defined in the abstract, 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 application's layered Java architecture has the following advantages, among others:

- 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, together with the explanations that follow, offer a high-level conceptual view of RPM's service-oriented architecture. 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.



Client

The application's client layer is comprised both of the GUI and interfaces. The GUI is responsible for presenting data to the user and for receiving 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 issues.

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

For more information about how RSL integrates with RPM, see Chapter 5, "Integration Methods and Communication Flow." For more information about batch processing, see Chapter 7, "Java Batch Processes."

Application Services Layer (Stateless Session Beans)

Application services are designed to provide specific services and specific data requirements to a particular client. What application services a client calls depends upon its needs and the data formats it has. Application services are concerned with somewhat narrow processes. Not surprisingly, the names of application services often correspond to client-related processes.

The application services layer of RPM's architecture implements the enterprise Java bean (EJB) type called stateless session beans (SSB). An SSB is a type of EJB that provides stateless service to a client. For example, a stateless session bean could be designed for the GUI. The application services reside on the server side of the process boundary (also known as the remote call boundary).

The application-specific services layer provides an interface between a particular client and the adjacent core services layer. To solve a business problem, application services call one or more core services. Application services could also call other application services. For example, one application service has a large granularity and needs another one to perform minor grain transformations.

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 application services layer's primary function is to facilitate the conversion of value objects/payloads to business objects and business objects to value objects/payloads which are required by the adjacent layers. The value objects/payloads accepted from and returned to the application services layer are nothing more than data-centric classes which 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 upon 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 the RPM application's core business logic. Core services are core in the sense that they work with the business object model, and they 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 close 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 that has a lifecycle has a state machine, which describes the object's lifecycle.

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 thus 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 an application's business related logic through one or more high-level methods. The core services represent all logical tasks that can be performed on an application's 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's business object 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 application's storage platform, containing the physical data used throughout the application. The system is designed to include two RDMS data sources, RPM and RMS.

Security

The embedded 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).

User Repository (Such As a Third-Party Directory Server)

To facilitate the authentication of users, RSM is integrated with a third party directory service application. Core services interact using Light Directory Access Protocol (LDAP), which allows RSM to talk with the third party directory service. The LDAP standard defines a network protocol for accessing information in a directory.

Although RSM is configurable to use any LDAP-compliant directory server, the system is certified to work with Oracle's Oracle Internet Directory (OID). The OID is an LDAPv3 directory that leverages the scalability, high availability and security features of the Oracle Database.

RSM uses LDAP for two purposes:

- As the master repository of user information
- As a third-party authentication service

In the second case, RSM authenticates users by binding to the LDAP Directory Server as the user who is attempting to log in to RSM. The user's password is never stored in RSM; it is passed along when RSM tries to connect to the Directory Server. If the connection to the directory server succeeds, the user is considered authenticated in RSM.

If RSM cannot connect to a directory server; the user is not able to log in.

Note: RSM never writes data to the LDAP Directory Server.

For additional information about Oracle Internet Directory (OID), see the following website: http://www.oracle.com/technology/products/oid/index.html.

Asynchronous 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 asynchronous conflict check process in RPM is a mechanism by which applications can execute long-running operations in the background while allowing work to be done simultaneously.

Within the RPM application, asynchronous processing is always utilized in the following functional areas:

- Price Changes/Clearances
- Promotions
- Worksheet

Asynchronous Processing Flow

The asynchronous conflict check process is performed as follows.

1. The client requests the application server to perform a process on a business object or set of business objects.
2. The server's application service layer extracts information about the request (type of business object, identifying ID, and requested action).
3. When RPM determines that a task is eligible for asynchronous processing:
   - A JMS message is published to the queue that contains specific information about the task.
   - A record is inserted into the TASK table. The task-specific information about the task is persisted as a Blob in the table. This task-specific information is a Java object that, when read from the database, is capable of calling an RPM application service to perform asynchronous processing and generate an alert when the processing is completed.
4. As soon as a message arrives in the queue, the container triggers execution of the RPMTaskMDB. RPMTaskMDB is a message-driven bean used to facilitate RPM's asynchronous processing capability. Message Driven Beans act as listeners to specified queues for messages.

5. The task passes information to the application service layer.
6. The application service layer performs a state transition on the requested business object (for example, approving a particular price change) based on the information passed by the task.
7. The results of this transition (for example, success or failure messages, conflict details) are recorded in the RPM_CONFLICT_CHECK_RESULT table.
8. The application service inserts a record into the ALERT table.
9. The client periodically polls the Alerts application service which looks for new alerts in the ALERTS table.

RPM Cached Objects

RPM can cache objects within the server in order 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 are not of specific criteria. An example would be to retrieve all clearance information for the Clearance Workflow. In contrast, parameterized database queries retrieve data that fall under a specific criteria such as retrieving 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. Further 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 they have not been 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 data stores that are affected by the transaction in a consistent state.
- Isolation: From the application's perspective, the current transaction is independent, in terms of application logic, from all other transactions running concurrently.
- Durability: The transaction's operations against a data store must persist.

Data access object (DAO)

This design pattern isolates data access and persistence logic. The rest of the component can thus 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 which 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 lower case 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 are performed by PL/SQL and are executed on the database server instead of the application server. As such, performance concerns around conflict checking functionality can be more efficiently addressed. To accommodate conflict checking, a number of PL/SQL package files must be installed on the database.

4

Conflict Checking

Conflict checking is made up of 18 rules that determine whether a price event can be approved or unapproved.

The rules are broken up into three categories:

- Merge Validator Conflict Checking rules

  These rules are "pre-merge," meaning that the rules are run before the effect of the new price event is added to the RPM_FUTURE_RETAIL table.

- 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 prior to running through these rules. If any of these rules is violated by the price event, all effects of the price event are removed from the RPM_FUTURE_RETAIL table.

- 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.

This chapter also explains how to add user-defined conflict checking rules. An overview of Bulk Conflict Checking and Chunk Conflict Checking (and their impact on performance) also is provided.

Note: RPM users can view the conflict status of price changes, clearances, and promotions (simple and threshold) through the user interface. Price events in conflict status also may be refreshed.

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. Single UOM for all item/locations in a fixed-price price event

   This rule ensures that, for a fixed-price price event, only one UOM exists across all item/locations in the price event if a user does not specify a UOM to use. If the system is unable to find a single, unique UOM to use, a conflict is generated and the user must enter a value before attempting to approve the price event again. If a single, unique UOM is found, the price event is updated to reflect that value.

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. Multiple promotions for a customer segment

   This rule ensures that only one promotion exists at any point for an item/location/customer type timeline.

6. Approving a promotion exclusion to an already active promotion

   This rule ensures that a user is not able to approve a promotion for an item/location when an active promotion level exclusion exists for that item/location.

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.

The RPM_CONFLICT_QUERY_CONTROL table provides the ability to add some configuration your conflict checking. 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 date in the RPM_FUTURE_RETAIL table.

- 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 modified on or added to RPM_FUTURE_RETAIL or RPM_CUST_SEGMENT_PROMO_FR.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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 promotions.

- 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).

- 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 also can be configured by changing the system option.

- 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 also can be configured by changing the system option.

- RPM_CC_PROM_COMP_CNT.VALIDATE

  An item/location can exist on more than one promotion at a time, but it is limited by a set of system options.

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

  If Multiple Promotions Ind = Y, the maximum number of promotions that an item/location can exist on must be less than or equal to the system option value (MAX_OVRLP_PROM_COMP_DETAIL). This count applies only to non-customer segment promotions.

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

- 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.

- RPM_CC_PROMO_FIXED_PRICE.VALIDATE

  One fixed-price promotion maximum (valid for Simple promotions).

  Only one fixed-price simple 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.

Rules Controlled by System Options

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

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

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

- 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.

- 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.

- System Option: Apply Promotion Change Type 1st.

  Indicates the type of promotion change to apply first. This option is used only after fixed price and non-secondary promotions are processed. Ranking is applied to the promotion header before promotion components are considered. Promotions are applied to an item/location based on ranking order. These rules apply to all promotions, including those created using the InjectorPriceEventBatch program.

  Note: For information, see the subsection called "Details" under "InjectorPriceEventBatch Batch Design" in Chapter 7, "Java Batch Processes."

  Ranking order is as follows:

  1. Fixed price promotions.
  2. Non-secondary (as opposed to secondary flagged promotions).
  3. System option: promotion apply order (amount off or percent off).
  4. Secondary flagged promotions (promotion header check box).

  Example:

  Promotion 1 has one component. P1C1 is Amount Off.

  Promotion 2 has two components. P2C1 is Percent Off, and P2C2 is Fixed Price.

  Both promotions are non secondary, and the Apply Promotion Change Type 1st system option is set to Amount Off.

  For this example, the order of application is as follows:

  1. Promotion 2, Component 2 (P2C2)
  2. Promotion 2, Component 1 (P2C1)
  3. Promotion 1, Component 1 (P1C1)

  Because Promotion 2 includes a Fixed Price component, it is ranked higher than Promotion 1. All components under Promotion 2 are applied first, followed by the component under Promotion 1.

  When applying Promotion 2, P2C2 (the Fixed Price component) is applied before P2C1. After all components under Promotion 2 are applied, the component under Promotion 1 is applied.

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 is 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 one input parameter of type VARCHAR2. 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)
                          I_price_event_type IN     VARCHAR2)
RETURN NUMBER IS


   L_program      VARCHAR2(61)  := 'CUSTOM_RULE.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
             NULL cs_ promo_fr_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)
 union all
      select price_event_id,
             NULL future_retail_id,
             cust_segment_promo_id cs_promo_fr_id
        from rpm_cust_segment_promo_fr_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 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',
                             rec.cs_promo_fr_id);
      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_program,
                                    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).

RPM_FUTURE_RETAIL

RPM provides retailers a forward looking view of all active and pending approved pricing events affecting an item at a given location. To do this, RPM uses a set of three Future Retail tables. Although each of these tables serves a different purpose, all of them work together, as shown in the following:

Table Name	Description
RPM_FUTURE_RETAIL	This table serves as the main/"parent".
RPM_PROMO_ITEM_LOC_EXPL	This table is an item/location level representation of all active/approved promotions and serves as detail/"child."
RPM_CUST_SEGMENT_PROMO_FR	This table holds promotional retails specific to customer segments and serves as detail/"child."

The tables above are used and updated during the conflict checking process. See the *Oracle Retail Price Management Data Model* for the detail table definition of these tables.

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

RPM utilizes bulk processing for conflict checking of price events whenever possible.

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

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 table above, Zone 1 contains the following locations:

Location 1
Location 2
Location 3
Location 4

Zone 2 contains the following locations:

Location 5
Location 6
Location 7
Location 8
Location 9

The first process is to separate these price changes into several collections (referred to as sequence) based on the item/location combination so that there are no item/locations overlapping in the sequence. In the example above, the PC1 and PC2 are overlapping. This collection of price changes is then divided into two sequences:

Sequence	Price Change	Effective Date	Item / Location
1	PC 1 PC 3 PC 4	9/7/07 9/15/07 9/30/07	Item 1 / Zone 1 Item 2 / Zone 2 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 is no database locking issue. Note that for the overlapping price changes (PC1 and PC2), the decision of which price changes should go to the first or second sequence is based on the effective date of the price change.

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

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

In the example of the price changes above, if the THREAD_LUW_COUNT is set to 10 then the sequences are divided into threads similar to the following table.

Processed First: Sequence One:

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

Processed Second: Sequence Two:

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

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

Chunk Conflict Checking

RPM also can utilize chunk processing for conflict checking of price events whenever possible. This can be best illustrated by considering a promotion set up for a department with 10,000 items and a zone with five locations.

The above mentioned bulk conflict checking process would be able to have only one thread that could process all 50,000 item/locations involved with one promotion as suggested above. By chunking those 50,000 item/locations into smaller groupings, multiple threads can be utilized to execute the conflict checking process.

To determine if a price even should be processed through chunking, there is a row on the RPM_BATCH_CONTROL table with PROGRAM_NAME column equal to com.retek.rpm.app.bulkcc.service.BulkConflictCheckAppService.

- The value of the THREAD_LUW_COUNT column of a row on the RPM_BATCH_CONTROL table with PROGRAM_NAME column equal to com.retek.rpm.app.bulkcc.service.ChunkConflictCheckAppService.
- The value of the CONFLICT_CHECK_CHUNK_FACTOR column of a single row in the RPM_SYSTEM_OPTIONS table. This column need to be updated by a DBA from the back-end as there is no UI screen available to update this column.

The conflict checking process utilizes chunking if the number of item/locations for a price event is greater than or equal to the THREAD_LUW_COUNT times NVL(CONFLICT_CHECK_CHUNK_FACTOR, 2.5).

RPM then uses the value of the THREAD_LUW_COUNT column of a row on the RPM_BATCH_CONTROL table with PROGRAM_NAME column equal to com.retek.rpm.app.bulkcc.service.ChunkConflictCheckAppService as the chunk size. If this is NULL, the chunk size will be 50,000.

Skipping Conflict Checking during Submit

Normally during the submit process, RPM performs a conflict checking process to validate whether any of the item/location combinations affected by the price event has any conflict. If there is any conflict, the price event stays in Worksheet status and the conflict will be displayed to the user. Users have the option to skip the conflict checking process during the submit process, if they do not need that for their business process. They can do this by checking the Do not run Conflict Check for Submit system option check box. If they do this, RPM does not perform the conflict checking process (and will not show conflicts if there are any) when the user performs a price event submit action. The price event status is switched to Submit immediately.

Skipping Conflict Checking during Complex Promotion Approval

Users can skip conflict checking process during Complex Promotion approval (and un-approval). When users approve (or un-approve) a Complex promotion, the system does not update the Future Retail table and does not perform conflict validation, but it populates the Payload tables so that RPM can communicate this Complex promotion to other systems (for example, through RIB or an ORPOS extract).

If this is required, and it meets their business requirements, users can do this by checking the Do not run Conflict Check for Complex Promotion Approval system option check box. If this system option is checked, RPM does not perform the conflict checking process (and will not show conflicts if there are any) when users approve (or un-approve) a Complex promotion.

This option potentially improves the approval of very large Complex promotions (such as Dept/Zone Level) dramatically. But retailers must consider this very carefully, because the following functionality is lost:

- Overlapping promotions: Item/Location in the Complex promotion is not considered when validating the overlapping promotions.
- Price Change Promotion Overlap/Clearance Promotion Overlap: Item/Location in the Complex promotion is not considered when validating these types of overlapping.
- The Complex promotion information is not available in the Future Retail tables.
- Deal cannot be attached to Complex promotions.

Although users can change this system option from one state to another in the System Options Edit screen at any time, flipping the system option (while having active or pending Complex promotions in the system) could produce unpredictable results, such as the following scenarios:

- The Do not run Conflict Check for Complex Promotion Approval system option is checked. The user approves a Complex promotion, unchecks the system option, and un-approves the Complex promotion (or cancels/changes the end date after the promotion becomes active). RPM attempts to execute the conflict checking process but cannot find the related records in the Future Retail tables. The Complex promotion might fail un-approval.
- The Do not run Conflict Check for Complex Promotion Approval system option is unchecked. The user approves a Complex promotion. RPM executes the conflict checking process and updates the Future Retail tables. The user checks the system option and un-approves the Complex promotion (or cancels/changes the end date after the promotion becomes active). RPM un-approves (or cancels/changes the end date of) the promotion without executing conflict checking process. Related records in Future Retail tables are not updated.
- The Do not run Conflict Check for Multi-Buy Promotion Approval system option is checked. The user approves a Complex promotion, unchecks the system option, and approves another Complex promotion that is overlapping with the first promotion. During the overlapping constraint validation, RPM will not see the first promotion.

Conflict Checking Refresh

Within the RPM user interface, price events in confilct status can be refreshed. On the detail dialogs for price changes, clearances, and promotions (simple and threshold), a button is enabled when price events are in conflict status. When this button is clicked, a separate pop-up window opens to list all price events in conflict checking status. From this window, the user can click **Refresh** to make a call to the database. If the status of the price event has changed, the change is reflected in this window.

From this window, the user can click **Cancel** (to not update the price event dialog) or **Exit** (to refresh the price event screen with the updated status for price events). The conflict review list pop-up includes promotion component and promotion component detail ID, as does the conflict review task pane.

For information, see the *Oracle Retail Price Management User Guide.*

5

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's 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

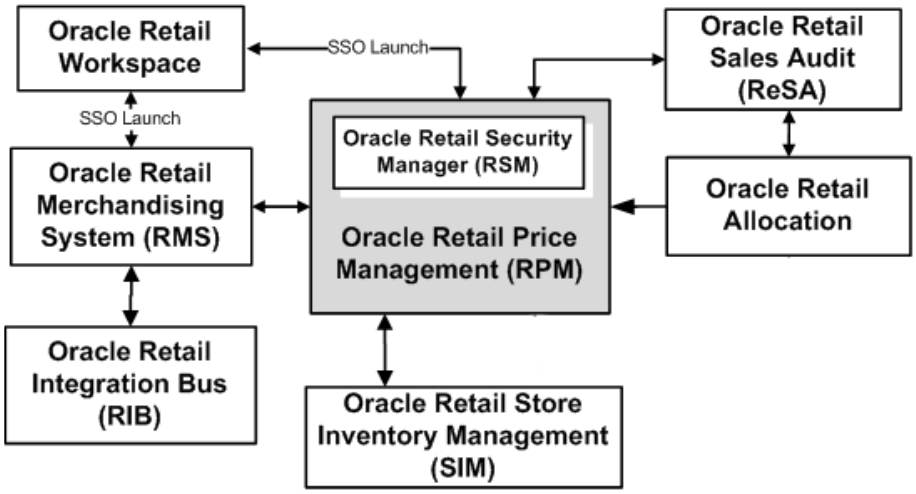
The following diagram details the overall direction of the dataflow among the various systems. The accompanying explanations of this diagram are written from a system-to-system perspective, illustrating the movement of data throughout the RPM-related portion of the enterprise. Note that this discussion focuses on a high-level functional use of data. A technical description of the dataflow is provided 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



Integration Interface Dataflow Description

The following describes the elements of the integration interface dataflow.

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: if there 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 are not applied to item/locations in RMS with a status of Deleted.

- 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.

Note: RMS does not notify RPM through RIB messages; rather, RMS communicates the information through database API and triggers. The activities that are communicated include creation of stores/warehouses, items/locations, and departments. Item modification also is communicated.

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, reason code and 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. Multi-buy 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 don't 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.

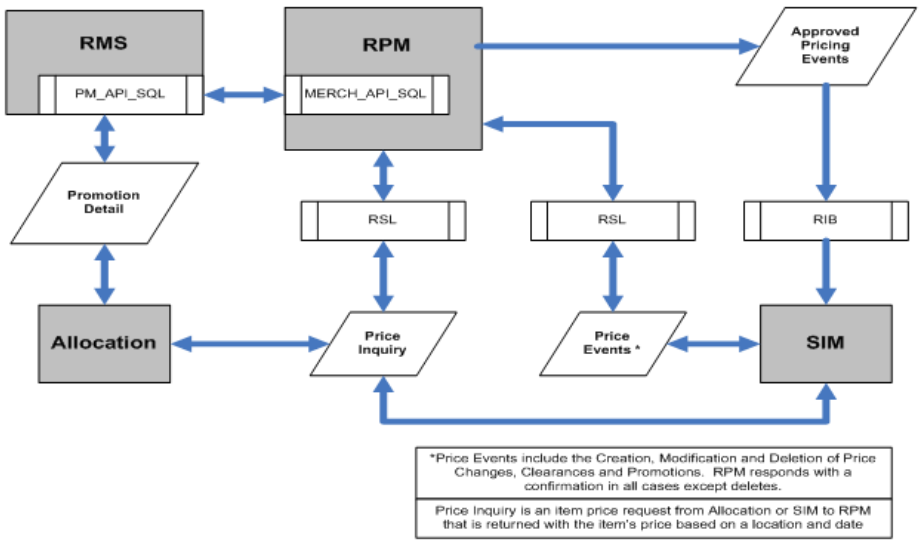
From RPM to the RIB and from the RIB to RPM

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. Multi-buy 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.

Pricing Communication Flow Diagram

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

Price Inquiry

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

Promotion Detail

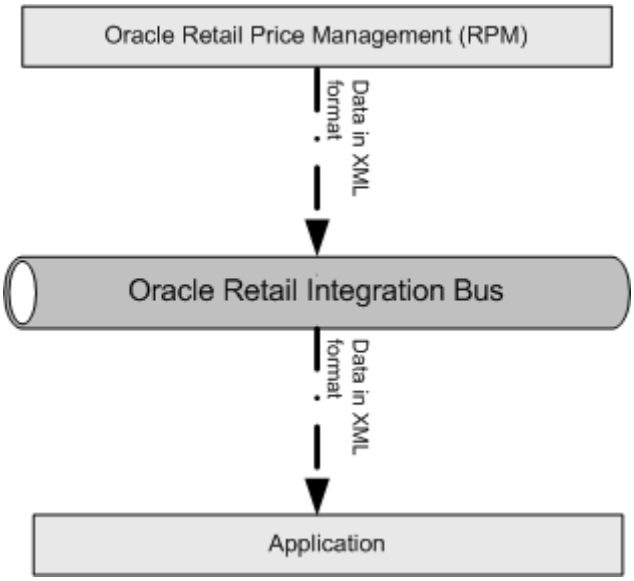
When Oracle Retail Allocation requires promotion detail, it is able to retrieve the data through RMS from RPM. There are two direct package calls involved - Oracle Retail Allocation calling the RMS package, and the RMS package calling an RPM package. This provides Oracle Retail Allocation with the promotional 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 "rib_user.properties" and"Disabling RIB Publishing in RPM" in Chapter 2, "Backend System Administration and Configuration".

The XML Message Format

As shown in the following diagram, the messages that RPM publishes are in an XML format; the data structure is defined by XML Schema Definitions (XSDs).



Message Publication Processing

As shown by the following diagram, an event within RPM's 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.

Publishers Mapping Table

This table illustrates the relationship among the message family, message type and XSD/payload. For additional information, see the latest Oracle Retail integration documentation.

Family	Type	XSD/Payload
regprcchg	REGPRCCHGCRE	RegPrcChgDtl
regprcchg	REGPRCCHGMOD	RegPrcChgDtl
regprcchg	REGPRCCGDEL	RegPrcChgDtlRef
clrprcchg	CLRPRCCHGCR	ClrPrcChgDtl

Family	Type	XSD/Payload
clrprcchg	CLRPRCCHGMOD	ClrPrcChgDtl
clrprcchg	CLRPRCCHGDEL	ClrPrcChgDtlRef
prmprcchg	MULTIBUYPROMOCRE	PromotionDesc
prmprcchg	MULTIBUYPROMOMOD	PromotionDesc
prmprcchg	MULTIBUYPROMODEL	PromotionRef

Functional Descriptions of Publication Messages

The following table briefly describes the functional role of publication messages in RPM functionality and the products involved with RIB integration. For information, see the Oracle Retail Integration Bus documentation set.

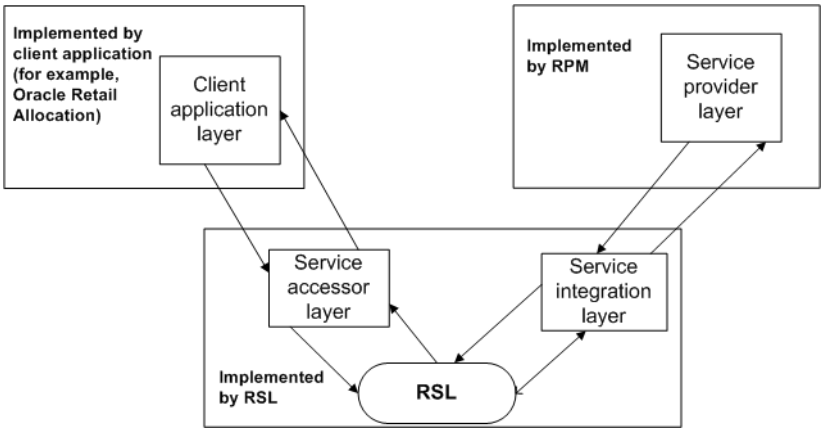
Functional Area	Integration to products	Description
Price Change Creation	RMS, SIM	This message is used by RPM to communicate the approval of a price change within the application. This message is published at a transaction item/location level.
Price Change Modification	RMS, SIM	This message is used 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.
Price Change Deletion	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	SIM	These messages are used by RPM to communicate the approval of a clearance price change or a clearance price change reset 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 through the effective date for the clearance price change reset.
Clearance Modification	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	SIM	This message is used by RPM to communicate the deletion (un-approval included) of an already approved clearance price change and any associated clearance price change resets. 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	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	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 promotion.
Promotion Deletion	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 following diagram, 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."



Functional Description of the Class Using RSL

The following table briefly describes the functional role that RPM's RSL class has within the application.

Class	Description
PriceChangeServiceJava.java	This service, provided by RPM, allows external systems such as SIM to create/modify/delete price events.
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 data sources, 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 following tables showing through the persistence layer:

RMS tables accessed through the persistence layer
AREA
CHAIN
CLASS
CODE_DETAIL
CODE_HEAD
COMP_PRICE_HIST
COMP_STORE
COMPETITOR
CUSTOMER_SEGMENTS
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
GTAX_ITEM_ROLLUP
ITEM_LOC
ITEM_MASTER
ITEM_SEASONS
ITEM_SUPPLIER
LOC_LIST_DETAIL
LOC_LIST_HEAD
PARTNER

RMS tables accessed through the persistence layer
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
VAT_ITEM
WH

RMS Packages and Methods Accessed through RPM's Persistence Layer

RPM uses the packages and methods shown in the following table 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 Views	RMS Tables
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
MERCH_NOTIFY_API_SQL

Oracle Retail POS Suite - RPM Integration

Integration between RPM and Oracle Retail POS Suite is optional. If you are not integrating RPM with Oracle Retail POS Suite, you can skip this section.

Overview

Oracle Retail POS Suite

RPM integrates with Oracle Retail POS Suite. Applications within Oracle Retail POS Suite include the following and more:

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

Integration

This section provides an overview of how RPM is integrated with Oracle Retail POS Suite.

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 POS Suites, see the *Oracle Retail POS Suite 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 (clearance and regular price changes)
- Price Promotion (simple promotions)
- Discount Rules (Threshold and Multi-buy promotions)
- Finance 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 Chapter 7, "Java Batch Processes," for information on batches.

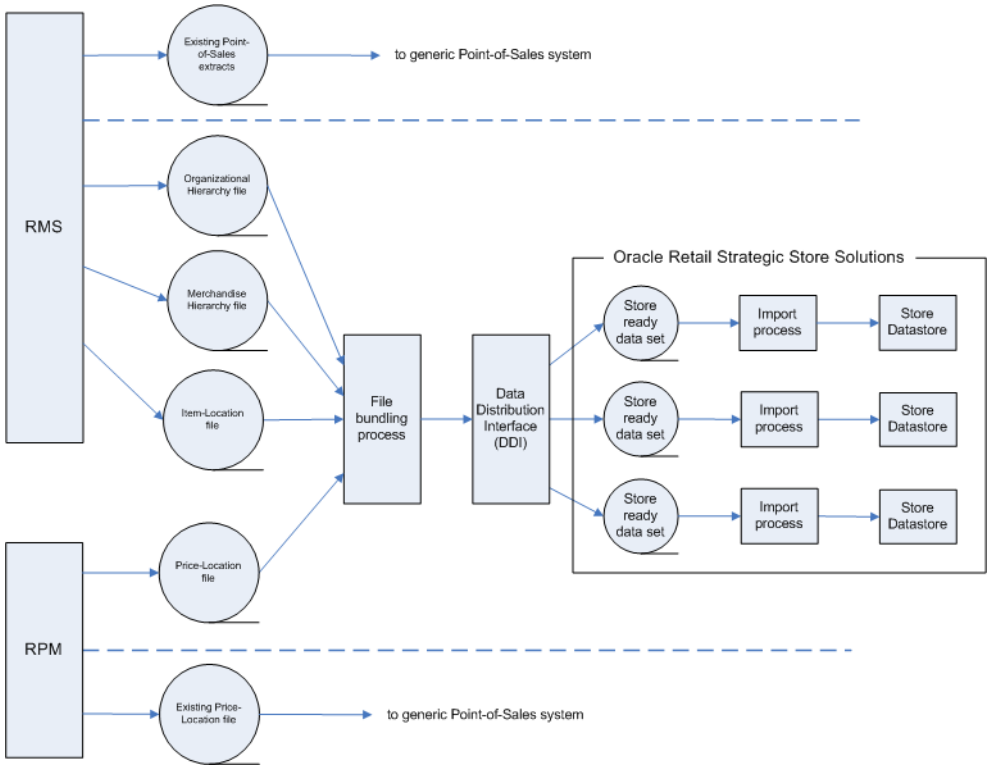
File Details

Several files are distributed to stores, as the following describes.

- Price: A store-specific file containing price changes, clearances, simple promotions, threshold promotions, and multi-buy promotions.
- New Item: A classification file, which is not store specific.
- Finance (Credit) Promotions: A store-specific file containing only finance promotions.

Integration Dataflow

The following diagram illustrates the overall direction of the dataflow among the various systems.



Functional Description of Dataflow

The following explanations, based on the diagram above, are written from a system-to-system perspective, indicating the movement of data throughout the RPM related portion of the enterprise.

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 sent to ORBO.

RPM creates the following data files for ORBO:

File	Description	Full Load or Incremental
Price	A store specific file containing price changes, clearances, and promotions.	Incremental
New Item	A classification file that contains non store specific information.	Full Load
Finance (Credit) Promotions	A store specific file containing only finance promotions	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 RPM to Oracle Retail POS Suite integration is done by jarring the XML files generated by SQL extract scripts.

Known Issues From the Oracle Retail Price Management Perspective

Mismatch in Promotion Functionality

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

The following items have been excluded from the XML:

- Threshold promotions with Qualification Type = “Item Level”.
- Multi-Buy promotions that combine quantity and amount requirements (buy 2 of X and spend at least $25 on Y, get $Z off).
- Multi-Buy promotions with no buy lists (produced by selecting reward type “Each Buy List”).
- Promotion details with change types Exclude or No Change.
- Promotion items for which an exception has also been approved.

- Amount threshold promotions containing a single item.
- Finance Promotions do not require an end date, but the associated field in CreditPromotion.xsd is required. RPM will not send a value for this field if the finance promotion was not created with a value for the end date.

Assumptions

- The phrase, Support the integration of Threshold promotions with a threshold Qualification Type =Item, is not currently supported by ORPOS and therefore is not included in this design.
- The phrase, Support the integration of the item's UOM as part of a promotion, is not currently supported by ORPOS and therefore is not included in this design.
- Although the ORPOS XSD specifies a five-character limit for store IDs, RPM sends the full store ID available from the database regardless of length. RPM allows for store IDs of up to 10 characters.

Other Gaps Between RPM and Oracle Retail POS Suite

- While multi-unit pricing can be set up in RPM it is not part of 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 POS Suite on the Pricing extract file.
- RPM clearance price changes are treated the same as regular price changes as Oracle Retail POS Suite does not distinguish the clearance price change type.

Known Issues From the Oracle Retail POS Suite Perspective

The following are known issues in functionality between Oracle Retail Price Management and Oracle Retail POS Suite, as encountered from the Oracle Retail POS Suite application perspective.

Functionality Gaps for Promotion Data Import

Identified Functionality Gap	Suggested Resolution
Oracle Retail Price Management supports a larger field (Change Value - Number) than does POS Suite. This field is the amount, either monetary or percent, to be used to change or replace the current selling price for a sale unit of an item. Could result in loss of data in case of a very large discount amount.	Gap to remain unchanged for this release.
In Oracle Retail Price Management, all applicable price promotions are applied. In Point-of-Service, if price promotion and discount rule apply to the same item, then the best deal is applied. If price change and discount rule or price promo apply to the same item, then both price change and promo or discount rule are applied	Oracle Retail Price Management turns off overlapping promotions. This ensures that only one promotion is applied to an item or location at a time.
The Item Number field is larger in Oracle Retail Price Management thanPOS Suite.	POS Suite logs an error if the database field is exceeded.

Identified Functionality Gap	Suggested Resolution
Field for Promotion Price attribute is larger in Oracle Retail Price Management. Multiple promotions can be applied, and the selling price represents the results of each promotion applied in the Apply Order. One record is downloaded for each promotion applied, and each has the same selling price. The stores system only applies the best deal, and it does so at the time the transaction is rung up. In addition to the multiple promotions, Oracle Retail Price Management can also apply price guides, which might specify the price ends in .99, for example. These price guides are not included in the download file. The selling price is ignored by Point-of-Service. This results in a possible problem if Point-of-Servcie does not calculate the same price that Oracle Retail Price Management sends as selling price. This discrepancy can result from rounding, price guides, and so forth.	Oracle Retail Price Management turns off overlapping promotions. This ensures that only one promotion is applied to an item or location at a time.

Functionality Gaps for Price Change Data Import

Identified Functionality Gap	Suggested Resolution
Oracle Retail Price Management supports a longer field (Selling Retail) and more precision.	Gap to remain unchanged for this release.
Oracle Retail Price Management Item field is longer.	Item ID length remains the same inPOS Suite and Oracle Retail Price Management. If the item ID is too long in the download file, the record is logged and discarded.
Oracle Retail Price Management does not support description field in download file.	Optional Description field is not populated.

Functionality Gaps for Discount Rule Data Import

Identified Functionality Gap	Suggested Resolution
Oracle Retail Price Management Item field length is longer.	Item ID length remains the same in POS Suite and Oracle Retail Price Management. If the item ID is too long in the download file, the record is logged and discarded.
Oracle Retail Price Management field (Threshold Value) is longer and supports more precision.	Field length remains the same in Oracle Retail Price Management andPOS Suite. If the threshold is a decimal value, it is logged and discarded.

Identified Functionality Gap	Suggested Resolution
Oracle Retail Price Management supports larger values and more precision than stores. Meaning of value (%, $, or new price) is defined by Change Type.	Field length remains the same in Oracle Retail Price Management and POS Suite.
Oracle Retail Price Management does not support threshold or limit.	Assume no threshold.
Oracle Retail Price Management does not support the Number Of Times Per Transaction (NbrTimesPerTrans) field.	Assume -1, which means no limit to the number of times the promotion can be applied to a transaction. The NbrTimesPerTrans attribute is in the PricingImport.xsd file.
Oracle Retail Price Management does not support the Accounting Method field.	Assume the discount.
Oracle Retail Price Management does not directly support the Allow Source to Repeat field.	Allow source to repeat.
Oracle Retail Price Management does not directly support the Deal Distribution field.	Assume target only.
Target Quantity field is not supported in Oracle Retail Price Management.	Assume target quantity of 1.

Integration with Oracle Retail Workspace

For information about Oracle Single Sign-On and Oracle Retail Price Management (RPM) 13.2.4, see the latest *Oracle Retail Price Management Installation Guide*.

For information on how to integrate RPM with Oracle Retail Workspace, see the latest *Oracle Retail Workspace Implementation Guide*.

Oracle Retail Workspace is a supported configuration of Oracle WebCenter Spaces 11.1.1.5 for Oracle Retail. For the Oracle Retail 13.2.x enterprise, Oracle Retail Workspace replaces previous versions of Oracle Retail Workspace. There is no more packaged Oracle Retail Workspace code, only configuration instructions for Oracle WebCenter Spaces 11.1.1.5.

The Oracle Retail Workspace configuration utilizes the external application functionality and the application navigator taskflow of the Oracle WebCenter Framework to configure RPM in Oracle WebCenter Spaces.

6

Functional Design

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.
- RPM uses RMS's VDATE to represent today's date rather than the server's system date.
- RPM only recognizes 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). If using GTAX in RMS, GTAX Item Rollup data must be in place in the merchandising system.
- Link codes are only used for regular price changes and are considered a point-in-time price change creation. If the link code relationship is updated, future price changes are not dynamically updated to reflect the change.

Functional Overviews

The following are overviews of RPM functionality.

Zone Structures

Zone structures in RPM allow you 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 these zone groups may be flexibly defined, they are primarily defined by their 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 items 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. These zones may be flexibly defined. For example, you may choose to create zones based on geographic regions such as the following:

- US East region
- US West region
- Mexico stores

Similarly, you 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.

- 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.
- 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 and deletes all future retail data for that zone/location.

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.

Users can add, modify, or delete the primary zone group definition for a given merchandise hierarchy within RPM. There are limitations for deleting a primary zone group definition . See the *Oracle Retail Price Management User Guide* for details.

If a user modifies an existing primary zone group definition or adds a new primary zone group definition at the merchandise hierarchy level that is lower than an already existing merchandise hierarchy (for example, a definition already exists for Dept 100 and a user creates a new definition for Dept 100 and Class 200), the primaryZoneModificationsBatch process must be executed for the changes to be reflected in the future retail tables. This batch requires that no other processes access the future retail tables as it runs. Also, the data in the merchandise hierarchy affected by the new or updated primary zone group definition should not be used until this batch has processed completely. Therefore, updating existing primary zone group definitions and adding new primary zone group definitions at the end of the day just prior to the batch window is highly recommended.

Codes

Market Basket Codes

A market basket code is a mechanism for grouping items within a hierarchy level in order 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.

Link codes are only used for regular price changes and are considered a point-in-time price change creation. If the link code relationship is updated, future price changes are not dynamically updated to reflect the change.

Price Changes, Promotions, Clearances, and Promotion Constraint

Within RPM, there are three primary categories of pricing events:

- Price changes
- Promotions
- Clearances

This section explains how price events are created. It also discusses how having access to margin information and historical promotion set-up data can help expedite price-event planning. Also included is 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.

Access to Current Margin Information

The RPM application receives near real-time cost changes from RMS. Therefore, when users create price events, they can see same-day cost changes that have occurred. This information allows users to make pricing decisions based on accurate cost and margin information. It is available through RPM Price Inquiry when planning price changes, clearance events and simple promotions.

The new margin information is based on the new retail and any cost changes made up to the time the change/promotion is created. When retail prices are tax-inclusive, the tax is stripped off the retail price when the margin is calculated. The resulting margin is correct only when the appropriate tax code/rate is considered in the calculation. In other words, the tax code/rate must be appropriate for the dates on which the price event is in effect.

The RPM application alerts users when a new, applied price event results in a negative margin. If the user chooses to go forward with it, RPM accepts the negative margin and displays it in bold--as long as it stays negative. Users also may see the effective date of the last cost change--as well as the last retail change to review--before approving a price event.

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, retailers specify the following:

- The item requiring 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 to which a price change can be applied is parent; the highest location level to which a price change can be applied is zone. Users have the option of creating exceptions to price changes created at one of these higher levels. For example, marking all men's turtleneck sweaters down 10%, but marking 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 go into effect, RMS pricing information is updated with the new regular retail resulting from the price change.

When creating fixed price changes, it is important that the selling UOM for the item matches the UOM for the item at its proposed price. Therefore, the system defaults proposed-price UOM to the selling UOM to reduce validation conflicts.

Promotions

Promotions are frequently driven by a particular event, such as a holiday or the overstock of an item. Promotions can be set up to apply to the regular retail price, the clearance retail price, or both--and when the promotion ends the price reverts back to the retail price that existed prior to the promotion.

When a promotion is entered in RPM, the following information is specified: The duration of the promotional price, what kind of promotion will take place, and to which items/locations the promotional price applies. The highest level at which a promotion can be set is department (for example, men's clothing). Users also have the

option of creating exceptions to promotions created at one of the higher levels. For example, marking all men's turtleneck sweaters down 10%, but marking down the large size 5%. Users also can exclude item/locations from a promotion.

Some examples of promotion types in RPM are:

- Complex promotion:
  - Meal Deal/Link Saver Promotion: For example, buy a sandwich, chips, and soft drink combination for $5.00, or get 25% off the total purchase.
  - Multi-Buy Promotion: For example, buy 3 of item A for $3.50 total.
  - Cheapest Item Promotion: For example, buy any three pairs of shoes from a list and get the least-expensive pair free.
- Simple promotion: For example, 25% off the retail price of an item.
- Threshold promotion: Spend $100, get $10 off.

  For all promotion types, RPM allows users to view setup information from past, completed promotions (within the last 12 months). This information can then be copied to create a new promotion.

  Note: The historical promotion data available for viewing and copying includes only setup data. Cost/margin and retail information for past promotions is not provided.

  When creating fixed promotions, it is important that the selling UOM for the item matches the UOM for the item at its proposed price. Therefore, the system defaults proposed-price UOM to the selling UOM to reduce validation conflicts.

Time-Based Promotions

Retailers need the ability to set up time-based promotions, such as Happy Hour and Door-Buster promotions. Users can enter start and end times, in hours:minutes, when entering start and end dates while setting up or maintaining promotions. Adding this data at the component-level detail allows for business validations, such as conflict checking. Once a promotion is approved, the information is sent to SIM and ORPOS. Start and end times can be added to the following promotions types, including customer segment and non-customer segment promotions:

- Simple promotions
- Threshold promotions
- Multi-buy promotions

You can also load time-based promotions using the price event injector batch. This functionality is supported for simple promotions only.

When creating promotions, the start and end times shown at header level will now be protected and defaulted to start time of 12:00 A.M. and end time of 11:59 P.M. Time based elements are only added at the component level. This allows users to create multiple components for a promotion with varying start and end times. There will be a new column that indicates time based promotion in the multi-records blocks for promotion search results, promotion component header, and the promotion component detail maintenance. The indicator will be checked if the promotion has any components that are set up as time based.

Time based elements for a promotion will be based on the time zone for the store location. A promotion that starts at 8:00 A.M. in New York will also start at 8:00 A.M. in California.

Users are also able to enter start and end times when creating emergency promotions. However, RPM is not designed to recognize time as it does for selecting promotion dates (based on business rules), making it possible that the start time for an emergency promotion could be in the past; there are no validation checks to prevent this scenario.

Example: User creates an emergency promotion to start at 8:00 A.M., ending at 1:00 P.M., but they don't enter the data into RPM until 9:30 A.M. The system will reflect the entered time values of 8:00 A.M. - 1:00 P.M., sending the data downstream to SIM and ORPOS.

Start and end times will default at the component detail level to 12:00 A.M. start and 11:59 P.M. end if user does not enter time values.

Price history in RMS only has the ability to reflect one price value for an item location and date. If multiple overlapping promotions exist for the same item location RMS will reflect the promotion with the latest start time. Because of these rules, it is possible that RMS and RPM current retail could be out of sync.

Example 1: An item location has two overlapping promotions:

- Early Bird promotion starting at 6:00 A.M. until 10:00 A.M.
- All day promotion starting at 8:00 A.M. until 10:00 P.M.

RMS will reflect the retails for the all day sale because it starts later than the Early Bird sale.

Example 2: An item location has two overlapping promotions only one is time based:

- Early Bird promotion starting at 6:00 A.M. until 10:00 A.M.
- Week long promotion - non time based 12:00 A.M. - 11:59 P.M.

RMS will reflect the retails for the Early Bird promotion (during the overlap day), because it starts later than the week long promotion.

The only discrepancy to this rule is when an emergency promotion is created. The emergency promotion will override any other existing promotions.

Note: Emergency promotions that are processed through bulk will follow this rule, if they are processed through chunk batch the updates are not applied until next day.

Canceling a Time Based Promotion

Users will have the ability to cancel a time based promotion. When an active promotion is canceled in RPM, the system will end the promotion at 11:59 P.M. on the current date (Vdate) regardless of when (what time of the day) the promotion was canceled. This process works the same as canceling a non-time based promotion. It is not realistic for store locations to support real-time promotion cancels as they would have to be able to react and update promotion signage in store.

Clearances

Clearances in RPM are defined as a markdown or a series of markdowns designed to increase demand and therefore move inventory out of a store. Subsequent clearances always result in the price of an item decreasing. When a clearance is created, you are

specifying the items 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.

Clearance Reset When a clearance price is created, you can specify a reset date on which the clearance price reverts back to regular retail price. Reset dates are optional.

For example, you can create a Clearance price event at the following levels: ParentItem/Zone, ParentItem/Location, ParentItemDiff/Zone, ParentItemDiff/Loc, TranItem/Zone, and TranItem/Location. When a Clearance price event is created at any of these levels, one record is created in the RPM_CLEARANCE table, where reset_ind = 0.

When the Clearance price event is approved, RPM generates additional Clearance Reset records in the RPM_CLEARANCE_RESET table at the TranItem/Location level for all TranItems/Locations affected by the Clearance, regardless of the level at which the Clearance is defined. For these Clearance Reset records, reset_ind = 1. If a Clearance Reset record is available for the TranItem/Location, only the existing Clearance Reset record for that TranITem/Location is updated.

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

ItemList Level Price Events

ItemList is a collection of transaction level items and/or parent items that is created/maintained in the Oracle Retail Merchandising System (RMS). RPM users can use the ItemList defined in RMS when creating a Price Change, a Clearance, or a Promotion.

When creating a Price Change, a Clearance, a Simple Promotion or a Threshold Promotion using the ItemList, users can choose to explode the ItemList into items (and create the price event at the item level) or create the price event at the ItemList level.

Whenever permitted by the retailer's business requirements, users should not explode the ItemList and create the price event at the ItemList level when using ItemList to create Price Change, Clearance, Simple Promotion or Threshold Promotion. The following advantages of doing this are explained.

If retailers use the ItemList that contains 1,000 items to create a Price Change, a Clearance, a Simple Promotion, or a Threshold Promotion, and the users decide to explode the ItemList, RPM will create 1,000 price events at the Item level. From a usability perspective, it is going to be a challenge for the user to maintain these 1,000 price event in the UI screen. Also, for a very large ItemList, it could cause some UI performance degradation.

If for the same scenario above, the user creates a price event at the ItemList level, RPM will create only one price event that will affect all 1,000 items in the ItemList. This is be much more manageable to maintain.

When creating an ItemList level price event, RPM uses the ItemList at a point in time. RPM captures the items in the ItemList during the creation of the price event i(when the price event is saved to the database) and uses this captured list of items throughout the lifetime of the price event. So after the price event is created and saved, adding or removing items to/from the ItemList (in RMS) will not impact the price event anymore.

As it is more beneficial for users to create price events at the ItemList level, a Allow Item List Explosion system option will decide whether users can explode the ItemList. If this system option is unchecked, the user will not have the option to explode the ItemList into items. So when the user selects an ItemList when creating a Simple Promotion, Threshold Promotion, Price Change, or Clearance, the price event is created at the ItemList level. If this system option is checked, the user can explode the ItemList into its items. If the user chooses to do that, RPM will create item level price events accordingly.

Promotion Constraint

Users are stopped if they are creating a price change within a set number of days of the start of an approved promotion or vice versa. Conflict checking stops the user from approving 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 sees a conflict in either the price change or promotion dialog and the price event is not approved. The user is able to optionally select to ignore promotion constraints on individual price change, promotion or worksheet detail record so promotion constraint checks are not performed when conflict checking is run.

Pricing Strategies

The pricing strategies front end allows you to define how item retails are proposed when pricing worksheets are generated. The strategies can be defined at department, class or subclass in order to represent which items are affected. The strategies are grouped into two categories, regular and clearance. An item/location can be on one maintain margin strategy and 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 it is applied to. The pricing strategy types are described in this functional overview and include the following:

- Area differentials
- Clearance
- Competitive
- Clearance Default
- Margin
- Maintain margin

When determining what merchandise level a pricing strategy is 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

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. Secondary area retails are generated when the primary area retail is changed due to a competitive retail change or a margin strategy retail change. Secondary area proposed price changes can be created in worksheet status to allow for user approval or can be dynamically updated to allow the user to work through multiple zone worksheets at the same time (primary zone included). If dynamic update is chosen, the system provides a dynamic update of the worksheet changes from the primary area to the secondary area. Following the same logic as that which resides in a 'what if' scenario, price changes are not actually created in the worksheet dialog until the user approves a worksheet. A system option signifies whether this process is used.

Layered Competitive Pricing Comparisons in the Worksheet

In the worksheet, competitive pricing comparisons are layers 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 retail proposed 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 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 you 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, and the strategy also details 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

Clearance Default Strategy

This strategy allows the user to specify clearance defaults as low as the subclass level. Pricing Worksheets are not generated for this pricing strategy. The markdown defaults (or subsequent markdown cadence) is set up in the pricing strategy dialog and then applied in the manual clearance dialog. When the user creates a manual clearance, they need to click Apply subclass defaults in order for the subsequent markdowns to be automatically generated. Once they click Apply the user sees the clearance they manually created as well as clearance IDs for subsequent markdowns generated from the clearance default strategy. The subsequent markdowns that are created can be updated to have reset dates and out of stock dates. Because worksheets are not generated for this new strategy, a calendar is not attached.

Competitive Strategy

This strategy allows you 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 you to define the target amount of markup you want to have above the 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 the retails that are 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 are dependant 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 that 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's retail and the cost, or the market basket code margin associated with the item.

The user also has the ability to 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 through the worksheet once a price change proposal has been created.

Merch Extract Calculations

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

Market Basket Margin

This section includes calculations for GTAX and SVAT.

- GTAX Calculation

  The following is a calculation of proposed retail in margin strategy and maintain margin - margin basket code.

  1. Calculate the retail by applying margin target.

     Proposed retail exclusive of tax = basis cost/(1-margin target%) --retail markup

     or

     Proposed retail exclusive of tax = base cost*(1+margin target%) --cost markup.

2. Apply the taxes to calculate the proposed retail.

   Proposed retail (tax inclusive) = (proposed retail exclusive of tax/(1-consolidated tax rate))+consolidated tax value.

- SVAT Calculation

  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

This section includes calculations for GTAX and SVAT.

- GTAX Calculation

  The following is a calculation of proposed retail in maintain margin strategy - current margin.

  1. Calculate the current margin.

     Step 1: Calculate the retail minus taxes: Retail exclusive of tax on review period start date=retail inclusive of tax on review period start date-((Retail inclusive of tax on review period start date-consolidated tax value)*consolidated tax rate) -consolidated tax value.

     Step 2: Calculate the current margin:

     Current margin % using retail method = (retail exclusive of tax on review period start date - cost on review period start date) / retail exclusive of tax on review period start date.

     or

     Current margin % using Cost method = (retail exclusive of tax on review period start date - cost review period start date) / cost on review period start date.

  2. Calculate the proposed retail based on the current margin calculated above.

     Step 1: Apply the margin calculated above with the basis cost.

     Proposed retail exclusive of tax(retail markup) = (basis cost/(1-current margin%)

     or

     Proposed retail exclusive of tax(cost markup) = basis cost * (1+current margin%)

     Step 2: Apply the taxes to calculate the proposed retail

     Proposed retail (tax inclusive) = (proposed retail exclusive of tax value.

- SVAT Calculation

  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 includes 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	0.83	(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	0.82	(0.55/(1-.33))
	X		X	0.55	0.8	(0.55-0.5) + 0.75

Price Inquiry

Price inquiry is designed to allow 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. You can search for prices based on the following search criteria:

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

After the search criteria has been specified, item/location combinations are displayed with their corresponding dates, and the prices on those dates. Included in the display are the item/location regular, clearance, and promotional retails--as well as cost and margin information. Items can be retrieved at the parent, parent/differential, or transaction level, and valid locations are price zone or location. After you have retrieved the desired pricing information, you have the option of exporting the information outside of the system. For maximum performance, Price inquiry search limitations should be set in the system option dialog. 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 and 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 (with the exception of worksheet generation for the maintain margin strategy). Only item/zones with a proposed retail are populated in a worksheet generated from a maintain margin strategy.

The Worksheet dialog includes two main screen, Worksheet Status and Worksheet Detail. In the Worksheet Wtatus screen, a table is displayed, where each row represents an individual worksheet. Access to the records in the worksheet is dependent on worksheet status and your user role. Functions available include the following:

- Submit
- Approve
- Reject
- Reset
- Delete

Note: Actions are applied only to detail records in worksheets that allow that action. For example, a Submit action is not applied to a record in Approved status.

Among the information displayed is the proposed price change generated by the merchandise extract program and other information that can assist in determining whether to accept, reject, or modify proposed pricing in the worksheet. After these determinations have been made, you can submit the worksheet for approval, rejection, and so on.

A worksheet does not exist until the merchandise extract program has run. These worksheets then have a new status. 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 is responsible for creating worksheets based on calendars and pricing strategies.

This is a three-step process:

1. Identify the work to be processed.
2. Extract RMS data into RPM.
3. Use 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: This set of rules is 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 Yes, 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.

The following is a list of reasons why item/locations are not included in the worksheet:

- Any item that does not have a record on the future cost table for the location on a margin strategy.
- If there are varying link codes across the item/locations.
- If the strategy is set up at a zone level, and the unit of measure for the item varies across the locations in the zone.
- If 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.
- If the merchandise extract program in running against a strategy setup 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.
- If there is an exclusion candidate rule that is met.
- If the item is not ranged in the location or if the strategy is at the zone level and the item is not ranged to any location in that zone.
- Items are on the exclude list of an area differential strategy.

See Chapter 7, "Java Batch Processes," for information about this batch process.

Calendar

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

Once the time frame of the calendar has been established, you can specify review periods for the calendar, which is comprised of numbers of days. You can also specify the number of days between those review periods. Collectively, these completely 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 this calendar valid. The following is an example of a valid calendar:

Start Date: 01/01/09
End Date: 01/20/09

Review Period Duration: 3 Days
Days Between Review Periods: 3 Days

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

Aggregation Level

Aggregation level functionality is used in RPM to define parameters that vary at the department level. Within this functional area, you 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 you to select a location that exists in a zone and move to a different zone within a zone group on a scheduled date. The user chooses to approve the location move. A batch processes all approved location move records, run them for conflict checking and update them to scheduled status. The batch runs immediately before the Location Move Execution batch.

After conflict checking is complete, the process also allows the location to persist most valid pricing events through the move and to smoothly transition out of their old zone pricing strategies into the new zones' pricing strategies. System options provide the user the flexibility to configure location moves in the following ways:

- System option that specifies location move rules in regards to existing pending and active zone level promotions that a location is moving from or into. The overlap options are as follows:
  - Extend old zone's promotion and do not inherit new zones overlapping promotions: The location keeps running the old zones promotion. The location does not inherit any zone level promotion for the new zone if it overlaps the move date.

  - End old zone's promotion and inherit new zones overlapping promotions: The promotion ends at the location the evening before the location move date. The location inherits the new zones promotion that overlaps the move date, but the promotion only starts on the location move date.
  - Do not start old zone's overlapping promotions and inherit new zones overlapping promotions: The location does not start the promotion if the zone promotion overlaps the move date. The location inherits the new zones promotion that overlaps the move date and starts the same day the zone level promotion starts or starts the day the move is scheduled if the zone level promotion is already active.
- System options which determine whether the location should inherit the zone retail for the new zone it is moving into.
- System option to distinguish how location move validation handles pricing strategies with review periods that overlap a move date.

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

Application Security

This section describes Oracle application security, including permissions, roles, and users.

Oracle Retail Security Manager Overview

Oracle Retail Security Manager (RSM) is an application that provides a centralized method of authenticating and authorizing Oracle Retail system users. RSM leverages a Lightweight Directory Access Protocol (LDAP)-compliant directory service to authenticate valid users.

Named Permissions

One of the primary purposes of RSM is to establish who has access to what business functionality. To facilitate this processing, any application (Oracle Retail or non- Oracle Retail) that is utilizing RSM populates RSM tables with named permissions. These are pieces of business functionality around which the application has security. For example, if RPM has promotions functionality surrounded by security, RPM creates a promotions named permission. Named permissions data is sent to the RSM database during installation.

An RSM user could never change a named permission because the applicable outside application must respond to it. That is, once a user logs into an application (Oracle Retail or non- Oracle Retail), the application accesses RSM to request all the named permissions for this user. Within RSM, a user has a collection of roles, and roles have a collection of named permissions. For example, when the RPM user logs in, RSM provides the named permissions. RPM asks, Does this user have promotion capability? If the user does not have the capability, RPM does not display that functionality for the user.

Actions and Named Permissions

When each RSM-integrated application populates the RSM database with named permissions (during installation), the application specifies potential actions that are possible against a named permission. Named permissions contain flags that determine the following specific actions that can be taken in the system. For example, RPM might have a named permission script for Promotions that specifies the following for the actions:

- Edit: true
- View: true
- Approve: false
- Submit: false
- Emergency: false

The result of RPM's script would be that users in the RPM system could only be assigned view, edit or no action with respect to Promotions functionality.

Type of Action	Description
None	Users associated with the role have access to the permission but no actions.
Edit	Users associated with the role are allowed to create, update, and save any changes to a workflow.
View	Users associated with the role are allowed to see to all secured information in a workflow, but not make any changes to the data in the workflow.
Approve	Users associated with the role are allowed to change the status of a workflow to Approved.
Submit	Users associated with the role are allowed to change the status of a workflow from Worksheet to Submitted.
Emergency	Users associated with the role are granted special access that goes beyond normal day-to-day access to functionality. They can thus bypass normal delays in processing.

Content Models and Named Permissions

A content model defines in an xml document the task groups and tasks that are displayed in the task pad of an Oracle Retail GUI application window. By defining a content model and assigning named permissions to the content model attributes, applications can login to RSM and retrieve secure content in return. For example, an administrator can configure an application's content model to restrict certain tasks that are visible and/or editable by specific users. This is done by configuring named permissions in conjunction with content model tasks.

For RSM to manage another Oracle Retail application's content, the application's content model must be deployed with the RSM server. See the Oracle Retail application's documentation before modifying an application's named permission settings.

Hierarchy (Data Level) Permissions

RSM administers hierarchy (data level) permissions. To facilitate this functionality, any Oracle Retail application utilizing RSM for data level permissions populates RSM tables with its hierarchy types (that is, merchandise and location). Applications either provide the details of these types up front with SQL scripts or dynamically by implementing an RSM interface and exposing it to the RSM service. RSM does not understand application specific data (for example, RSM does not know the difference between departments and locations). To RSM, the data is a tag (for example, department) and a specific value (for example, 1000). This information is passed back to calling applications, and it is the applications responsibility to apply the data level permissions appropriately.

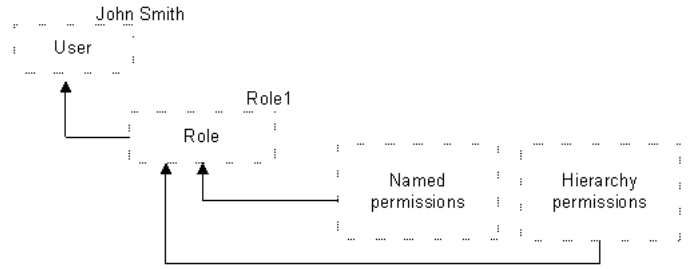
For example, when an RPM user logs in, RSM provides the hierarchy permissions for the user. RPM asks, Does this user have access to Department 1000? If the user does not have access, RPM does not display data from this department to the user. Like named permissions, within RSM a user has a collection of roles, and roles have a collection of hierarchy permissions.

Roles and Users

RSM allows for the creation of security roles. Roles consist of a unique identifier, an arbitrary name, and any number of permissions. Roles are created by the retailer and are used as a mechanism for administering its security requirements.

As the following diagram illustrates, roles are used as a mechanism for grouping any number of permissions. The role then is assigned to various users.

The security administrator assigns permissions to roles. To continue the earlier example, the security administrator could only assign a role with view or edit promotions functionality. Suppose that the security administrator decided to assign a role with view (a true flag behind the scenes) but not edit (a false flag behind the scenes), the security administrator could then assign a user, John Smith, to that role. John Smith could only view Promotions functionality.



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

Although this scenario only arises in a very specific case within a specific time window, ramifications of the resulting overwrites can be quite 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's processing. If a second user tries to lock the same data, he or she receives a message notifying them 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, you are not allowed to edit 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 enters into the detail screen. User two enters the worksheet status screen and decides to review the same worksheet. When user two selects and attempts to enter that worksheet, the system stops them. They are informed that user one is currently in the worksheet and they are not able to access it at this time.

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 the second user that they overwrote data modified by the first user.

If the second user's changes are incompatible with the first user's changes, business validation/rules will protect data integrity. In this case, the second user receives 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 their 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 their change. Since there is no validation that has been broken, the second user's change is also saved resulting in no difference from the first user. If the second user changed the value and validation failed, they are prompted with an error to fix the problem, just as though they created the validation error themselves.

Optimistic Data Locking

The second user receives an error message if they attempt to overwrite data modified by the first user. The message notifies the user that they have been working with stale data, so they should re-load and re-process their changes. With, Optimistic Data Locking, the first user wins; therefore, it is the opposite of the Last User Wins approach.

Concurrency Solution/Functional Area Matrix

Functional Area	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	

7

Java Batch Processes

This chapter discusses Java-based batch processing within RPM.

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 (such as batch return values)

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 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 RPM Batch Processes

This section describes Java-based and PL/SQL based batch processes.

Java Based Batch Processes

Java batch 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.

Also as part of the installation process, the application user and batch user alias are created and the corresponding credentials are stored inside a wallet. The batch user alias can be the same as the application user ID. RPM batches use the user alias provided at the command line to fetch the password from the wallet and authenticate the user permission before executing the batch. See the *Oracle Retail Price Management Installation Guide* for details. The batch shell scripts must be run from within the directory structure to which they are copied.

Each script performs the following internally:

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

To use the scripts, confirm that the scripts are executable (using ls –l) and run "chmod +x *.sh" if necessary. The shell scripts take at least one argument: userAlias and any extras necessary for the batch. The output can be redirected to a log file (as shown in the following example).

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 userAlias > log 2>&1
```

PL/SQL Based Batch Processes

PL/SQL batch 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 are copied to a client-specified directory structure. Also as part of the installation process, the database user alias is created and the corresponding credentials are stored inside a wallet. The database user alias can be the same as the application user ID. RPM batches use the user alias provided at the command line to fetch the database connection string from the wallet and authenticate the user as part of any database connections that happen as part of the batch. See the *Oracle Retail Price Management Installation Guide*for details. The batch shell scripts must be run from within the directory structure to which they are copied.

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. The one exception to this is the RPMtoORPOSPublishExport.sh batch. When it runs run on a SunOS system, it must be run with a Korn Shell interpreter.
- 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
clearancePriceChangePublishExport.sh	FutureRetailRollupBatch
FutureRetailRollUpBatch.sh	
GenerateFutureRetailRollUpBatch.sh	GenerateFutureRetailRollupBatch
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**: This script sets up environment information and takes as a parameter the name of the batch program to run.
locationMoveBatch.sh	LocationMoveBatch
locationMoveScheduleBatch.sh	
merchExtractKickOffBatch.sh	MerchExtractKickOffBatch
newItemLocationBatch.sh	NewItemLocationBatch
PriceChangeAreaDifferentialBatch.sh	PriceChangeAreaDifferentialBatch
priceChangeAutoApproveResultsPurgeBatch.sh	PriceChangeAutoApproveResultsPurgeBatch
priceChangePurgeBatch.sh	PriceChangePurgeBatch
priceChangePurgeWorkspaceBatch.sh	PriceChangePurgeWorkspaceBatch
priceEventExecutionBatch.sh	PriceEventExecutionBatch
priceEventExecutionDealsBatch.sh	
priceEventExecutionRMSBatch.sh	
priceStrategyCalendarBatch.sh	PriceStrategyCalendarBatch
primaryZoneModificationsBatch.sh	
processPendingChunkBatch.sh	ProcessPendingChunkBatch
promotionArchiveBatch.sh	PromotionArchiveBatch
promotionPriceChangePublishBatch.sh	PromotionPriceChangePublishBatch
promotionPriceChangePublishExport.sh	
promotionPurgeBatch.sh	PromotionPurgeBatch
purgeBulkConflictCheckArtifacts.sh	purgeBulkConflictCheckArtifacts
purgeExpiredExecutedOrApprovedClearancesBatch.sh	PurgeExpiredExecutedOrApprovedClearancesBatch
purgeLocationMovesBatch.sh	PurgeLocationMovesBatch
PurgePayloadsBatch.sh	PurgePayloadsBatch
purgeUnusedAndAbandonedClearancesBatch.sh	PurgeUnusedAndAbandonedClearancesBatch
refreshPosDataBatch.sh	
regularPriceChangePublishBatch.sh	RegularPriceChangePublishBatch

Script	Batch program executed
regularPriceChangePublishExport.sh	
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 through the shell scripts. These scripts take any and all arguments that their corresponding batch process would take when executing.

Functional Descriptions and Dependencies

The following table summarizes RPM's batch processes and describes the business functionality of each batch process.

Batch processes	Details
ClearancePriceChangePublishBatch	This batch process formats and stages output of clearance price change price events.
FutureRetailRollUpBatch.sh	This batch attempts to roll up timelines at a lower level by comparing lower level timelines to higher levels and removing any lower level timelines that match higher level timelines exactly.
GenerateFutureRetailRollUpBatch.sh	This batch is a conversion process that utilizes existing item/location level timeline data to create timelines at higher levels.
InjectorPriceEventBatch	This batch program performs the necessary work to import pricing events (regular price changes, clearance price changes and simple promotions) and optionally submit the events for approval.
ItemLocDeleteBatch	This batch program handles RMS deletions of item locations.
itemReclassBatch	When items are moved from one department/class/subclass to another in the merchandising system, this batch process accordingly sets the correct department/class/subclass for these items in the RPM_FUTURE_RETAIL table.
LocationMoveBatch	This batch process moves locations between zones in a zone group.
MerchExtractKickOffBatch	This batch process builds worksheets in RPM. MerchExtractKickOffBatch.java either creates or updates worksheets based on price strategies and the calendars attached to them.
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 through the RPM_STAGE_ITEM_LOC table, which is populated by an RMS process.

Batch processes	Details
PriceChangeAreaDifferentialBatch	This batch program allows the user to generate and approve Price Changes for selected secondary areas of an Area Differential.
PriceChangeAutoApproveResults PurgeBatch	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.
primaryZoneModificationsBatch.sh	This batch updates the future retail tables whenever the primary zone group definition for a merchandise hierarchy is updated, added, or removed so that the future retail tables reflect timelines at the zone level correctly.
ProcessPendingChunkBatch	This batch will attempt to re-process any push back threads that failed during the chunk conflict checking process.
PromotionArchiveBatch	This batch archives promotions that have an end date in the past.
PromotionPriceChangePublishBatch	This batch process formats and stages output of promotion price change price events.
PromotionPurgeBatch	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.
PurgeExpiredExecutedOrApproved ClearancesBatch	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.
PurgeUnusedAndAbandoned ClearancesBatch	This batch process deletes unused and rejected clearances.
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 the SQL script, RPMtoORPOSSpoolMsg.sql. This script spools the data collected from the publish tables of price events (Price Change, Clearance and Promotion). TheRPMtoORPOSPublishExport program also may call the RPMtoORPOSSpoolCP.sql script, based on an input parameter and the RPMtoORPOSSpoolMsg.sql script. This script spools credit promotion specific data from the publish tables.
statusPageCommandLine Application	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.

Batch processes	Details
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 Batch Schedule document published in conjunction with this release.

Threading and the RPM_BATCH_CONTROL Table

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

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

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

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

Return Value Batch Standards

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

Return Values

0 – The function completed without error.

1 – A fatal error occurred. The error messages are logged, and the process is halted.

Batch Logging

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

For more information, see Chapter 2, "Backend System Administration and Configuration."

ClearancePriceChangePublishBatch Batch Design

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 userAlias
```

The following command runs the clearancePriceChangePublishExport job:

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

Where the first argument is the database connect alias (/@db_connection_alias) and the second argument is the path where the file should be written. The path is optional and if not supplied, the path ../output is 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. After the flat file is successfully generated by the Export script (see following format), the associated records in the payload tables are deleted.

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 identification
	File Type	Char(5)	CLRPC	Clearance Price Changes
	Export timestamp	Timestamp		System clock timestamp (YYYYMMDDHHMISS)
FDETL	Record Descriptor	Char(5)	FDETL	File Detail Marker (1 per clearance create or modify)
	Line id	Number(10)		Unique line identification
	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

Record Name	Field Name	Field Type	Default Value	Description
	Selling Retail UOM	Char(4)		Selling retail unit of measure
	Selling Retail Currency	Char(3)		Selling retail currency
	Reset Clearance Id	Number(15)		Clearance reset identification
FDELE	Record Descriptor	Char(5)	FDELE	File Detail Delete Marker (1 per clearance delete)
	Line id	Number(10)		Unique line identification
	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 identification
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

Assumptions and Scheduling Notes

ClearancePriceChangePublishBatch should be run after the WorksheetAutoApproveBatch.

ClearancePriceChangePublishExport should be executed as follows:

- After every successful run of ClearancePriceChangePublishBatch.
- BeforePurgePayloadsBatch.

Primary Tables Involved

- RPM_PRICE_EVENT_PAYLOAD
- RPM_CLEARANCE_PAYLOAD

Threading

The ClearancePriceChangePublishBatch program 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
```

FutureRetailRollUpBatch Design

This batch attempts to roll up timelines at a lower level by comparing lower level timelines to higher levels and removing any lower level timelines that match higher level timelines exactly.

Usage

The following command runs the FutureRetailRollUpBatch job:

```
FutureRetailRollUpBatch userid password [dept=<deptId> class=<classId>
subclass=<subclassId>]
```

Where the first argument is the user ID, and the second argument is the password. The remaining arguments are optional, but the merchandise hierarchy to be processed can be specified.

Details

The batch calls the package RPM_ROLLUP.THREAD_ROLLUP_FR and RPM_ ROLLUP. ROLL_FUTURE_RETAIL. This packageattempts to roll up lower level timelines to existing higher level timelines (for example, from Item/Location to Parent/Zone) by comparing two related timelines and removing the lower level timelines if the two match exactly for all records.

Assumptions and Scheduling Notes

This process must be executed during the batch window. As it runs, other processes must not access the future retail tables.

Primary Tables Involved

- RPM_FUTURE_RETAIL
- RPM_PROMO_ITEM_LOC_EXPL
- RPM_CUST_SEGMENT_PROMO_FR

Threading

The FutureRetailRollUpBatch program is threaded by item.

GenerateFutureRetailRollUpBatch Design

This batch is a conversion process that utilizes existing item/location level timeline data and create timelines at higher levels.

Usage

The following command runs the GenerateFutureRetailRollUpBatch job:

```
GenerateFutureRetailRollUpBatch userid password [dept=<deptId> class=<classId>
subclass=<subclassId>]
```

Where the first argument is the user ID, and the second argument is the password. The remaining arguments are optional, but the merchandise hierarchy to be processed can be specified.

Details

The batch calls the package, RPM_GENERATE_ROLLUP_FR_SQL.THREAD_ROLLUP_FR and RPM_GENERATE_ROLLUP_FR_SQL. GENERATE_ROLLUP_FR. This package generates timelines above the Item/Location level using existing Item/Location level timelines as the following describes:

1. For every Parent/Zone combination within the MOM infrastructure, this conversion script gathers up all existing Item/Location timelines on Future Retail and counts the number of records in each timeline with price events affecting them (by looking at the price_change_id field, the clearance_id, on_simple_promo_ind and on_complex_promo_retail), along with the total number of records for each timeline.
2. With these two counts at hand, all timelines with the same count of records in their timelines are grouped together. For the sake of example, the group with the greatest number of timelines in it is Group A.
3. The timelines are sorted by the count of price events affecting that timeline. The timelines with the fewest price events affecting them are identified.
4. In the order specified here, the timeline in Group A with the fewest price events and the fewest location moves affecting it is selected. If there is more than one timeline that meets these requirements, the timeline with the Item ID that has the smallest value is selected. If more than one timeline meets these requirements, the timeline with the smallest Location ID value is selected from the remaining timelines.
5. For every Item/Zone combination within MOM, where the item has no parent item, the same logic is utilized, except the information is grouped initially by Item/Zone rather than by Parent/Zone.

6. For every Parent/Loc combination within MOM, the same logic is utilized, except the information is grouped initially by Parent/Location rather than by Parent/Zone. This is done for all locations, regardless of whether the location is part of the primary zone group for the given item's merchandise hierarchy.
7. For every Parent-Diff/Zone combination within MOM for diff_1 items above the transaction level, the same logic is utilized, except the information is grouped initially by Parent-Diff/Zone rather than by Parent/Zone.
8. For every Parent-Diff/Loc combination within MOM, for diff_1 items above the transaction level, the same logic is utilized, except the information is grouped initially by Parent-Diff/Location rather than by Parent/Zone. This is done for all locations, regardless of whether the location is part of the primary zone group for the given item's merchandise hierarchy.

The Item/location timelines that are selected to be copied for generating higher level timelines are not necessarily those that will allow the FutureRetailRollUpBatch process to reduce the data load as much as it possibly could. However, this process allows the system to remove lower level timelines in the future, as the future retail timelines will become more and more in line from higher to lower levels (for example,from Parent/Zone level to Item/Location level).

Assumptions and Scheduling Notes

This process has to be executed as part of a conversion/upgrade process.

Primary Tables Involved

- RPM_FUTURE_RETAIL
- RPM_PROMO_ITEM_LOC_EXPL
- RPM_CUST_SEGMENT_PROMO_FR

Threading

The GenerateFutureRetailRollUpBatch program is threaded by item.

InjectorPriceEventBatch Batch Design

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

Usage

Use the following command to run the job:

```
InjectorPriceEventBatch userAlias status=status_value event_type=event_type_value
polling_interval=polling_interval_value
```

Where:

- userAlias is a required argument. It represents the alias tied to a valid RPM user as stored in wallets.
- status_value is an optional argument. It defines the status for the imported data to process. Valid options are N (New), E (Error), W (Worksheet) or F (Failure). The default is N (New).

- event_type_value is an optional argument. It defines the type of pricing event to process. Valid options are PC (price change), CL (clearance) or SP (simple promo). The default is PC (price change).
- polling_interval_value is an optional argument. It defines the interval in seconds for the batch to verify if conflict checking is complete. Valid diapason for the interval is 1 to 1000. The default is 10 seconds.

UserAlias should be first in the list of arguments; all other optional arguments may follow in any order.

Except for userAlias, the argument identifier 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 InjectorPriceEventBatch job:

- Where only user alias is defined by the user

  Arguments:

  ```
  InjectorPriceEventBatch retailUser
  ```

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

- Where all arguments are defined by the user

  Arguments:

  ```
  InjectorPriceEventBatch retailUser 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).

Additional Notes

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

Details

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

Importing Staged Price Changes

Staged price changes data should be placed by the system administrator into RPM_STAGE_PRICE_CHANGE table. The table has a structure similar to that of RPM_PRICE_CHANGE but with the following 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 successfully importing the data.
- Status defines the current state of the data in the staging table. The status should not be confused with the status of the price event in RPM, even though there are some correlations. Possible statuses are N (New), W (Worksheet), A (Approved), E (Error) and F (Failed). Initial data should be created in the New status. The rest of the statuses are the result of data lifecycle.

  The records in all but Approved statuses are eligible for processing by the batch:

  – New status indicates the data has not been processed yet.

  – Worksheet status indicates the data has been processed and been successfully imported into RPM. It also indicates that at the time of import, approval was not required.

  – Error status indicates the data has been processed, but the import has failed due to invalid data. The data administrator can correct the data, and processing can be retried.

  – Failed status indicates the data has been successfully imported, but some conflicts have been encountered. The data administrator can correct the data (for example, by changing the effective date of the event to eliminate the conflict), and processing can be retried.

  – Approved status indicates the data has been successfully imported and successfully approved without any conflicts.
- The error message field indicates the first error encountered while importing data. The field actually holds an error message key rather than the actual error message. When a conflict is encountered during an approval attempt, the field holds the CONFLICT_EXISTS key.
- Process ID uniquely identifies the batch run. The field is populated for records with all statuses except New.
- 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 has a structure similar to that of RPM_CLEARANCE but with the following limitations:

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

As the PROMO_ID column in the RPM_STAGE_PROMO_COMP_SIMPLE table is a required column, this batch job does not generate the Promotion Header record and assumes the simple promotion will be injected into an existing Promotion Header. This can be achieved by using the existing Promotion Header or by populating the Promotion Header (RPM_PROMO table) before populating the RPM_STAGE_PROMO_COMP_SIMPLE staging table.

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

- Auto approve indicator defines if the processing batch should attempt to approve the clearance after successfully importing the data.
- Status defines the current state of the data. Possible statuses are N (New), W (Worksheet), A (Approved), E (Error) and F (Failed). Initial data should be created in the N (New) status. The rest of the statuses are the result of data lifecycle. The records in all but Approved statuses are eligible for processing.
  - New status indicates the data has not been processed yet.
  - Worksheet status indicates the data has been processed and been successfully imported into RPM. It also indicates that at the time of import, approval was not required.
  - Error status indicates the data has been processed, but the import has failed due to invalid data. The data administrator can correct the data, and processing can be retried.
  - Failed status indicates the data has been successfully imported, but some conflicts have been encountered. The data administrator can correct the data (for example, by changing the effective date of the event to eliminate the conflict), and processing can be retried.
  - Approved status indicates the data has been successfully imported and successfully approved without any conflicts.
- The error message field indicates the first error encountered while importing data. The field actually holds an error message key rather than the actual error message. When a conflict is encountered during an approval attempt, the field holds the CONFLICT_EXISTS key.
- Process ID uniquely identifies the batch run. The field is populated for records with all statuses except New.
- 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 has the structure similar to that of RPM_PROMO_COMP_SIMPLE but with the limitation that no exceptions are allowed.

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

- Auto approve indicator defines if the processing batch should attempt to approve the promotion after successfully importing the data.
- Status defines the current state of the data. Possible statuses are N (New), W (Worksheet), A (Approved), E (Error) and F (Failed). Initial data should be created in the N (New) status. The rest of the statuses are the result of data lifecycle. The records in all but Approved statuses are eligible for processing.
  - New status indicates the data has not been processed yet.
  - Worksheet status indicates the data has been processed and been successfully imported into RPM. It also indicates that at the time of import, approval was not required.

  - Error status indicates the data has been processed, but the import has failed due to invalid data. The data administrator can correct the data, and processing can be retried.
  - Failed status indicates the data has been successfully imported, but some conflicts have been encountered. The data administrator can correct the data (for example, by changing the effective date of the event to eliminate the conflict), and processing can be retried.
  - Approved status indicates the data has been successfully imported and successfully approved without any conflicts.
- The error message field indicates the first error encountered while importing the data. The field actually holds an error message key rather than the actual error message.
- Process ID uniquely identifies the batch run. The field is populated for records with all statuses except New.

Main Steps Taken by the Batch

- Generate the pricing events (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 is reported (ERROR_MESSAGE column in the staging table is populated), and the status is set to ERROR. The records with multiple incorrect data fields would need to be processed multiple times unless corrected by the data administrator all at once. If no pricing events have been generated the batch 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 do not impact processing of other records.
- If at least a single pricing event is generated the batch proceeds with an optional approval step. For the batch to attempt the approval on a record, the data administrator populating the staging data should set auto approval flag on the record to ON (AUTO_APPRVE_IND should be set to 1).

  In this case the batch attempts to approve the price event. This involves conflict checking, which can take quiet a long time depending on the volume of data. This step ends 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.
- If auto approval was not requested or approval process failed, the data is left in appropriate RPM table in Worksheet status. If the 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.
- 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 was determined that the CC logic is complete the batch purges CC intermediate data.

- 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:
  - The initial status in which the batch was processed
  - Event type
  - Number of records imported
  - Number of records requiring approval
  - Number of records successfully approved

Assumptions and Scheduling Notes

- It is assumed that a single instance of the batch is running at a time so a single event type is processed 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, RPM_PROMO_COMP, RPM_PROMO_DTL, RPM_PROMO_DTL_LIST_GRP, RPM_PROMO_DTL_LIST, RPM_PROMO_DTL_MERCH_NODE, RPM_PROMO_DTL_DISC_LADDER, RPM_PROMO_ZONE_LOCATION.

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

Threading

The InjectorPriceEventBatch program 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 polls 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.
3. Price events are created in the user interface and the staging table is updated with status of A (Approved).

4. 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 so that all of the price events are executed with a status of A. The parameter sets the price event back to Worksheet status.
5. 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 items and item locations. When RMS deletes an item or an item location, RPM removes the Item/Location rows from the RPM_FUTURE_RETAIL table so that pricing events are no longer published out of RPM. RPM will also remove any data that's specific to items and item/locations deleted from RMS in price event tables. This batch will also remove records from the RPM_ITEM_LOC table when an item/location relationship is removed.

The RPM_STAGE_DELETED_ITEM_LOC table is populated by the trigger RMS_TABLE_RPM_ITL_AIR.

Usage

The following command runs the ItemLocDeleteBatch job:

```
ItemLocDeleteBatch.sh userAlias
```

The argument is the user ID.

Follow these steps to use this batch:

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. 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

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 userAlias
```

Where the argument is the user ID.

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

The itemReclassBatch program is not threaded.

PL/SQL Interface Point

Package: RPM_ITEM_RECLASS

LocationMoveBatch Batch Design

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

Usage

The following command runs the LocationMoveBatch job:

```
LocationMoveBatch userAlias[max_retry]
```

Where the first argument is the userAlias. UserAlias is a required argument. The second argument (max_retry) is optional and specifies the maximum retry count for the batch.

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

- RPM_LOC_MOVE_PRC_CHNG_EX
- RPM_LOC_MOVE_CLEARANCE_EX
- RPM_LOC_MOVE_PROMO_COMP_DTL_EX

Update the status of the location move to Executed.

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

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

Assumptions and Scheduling Notes

LocationMoveBatch must run before the following programs:

- PriceEventExecutionBatch
- MerchExtractKickOffBatch

Primary Tables Involved

RPM_FUTURE_RETAIL

Threading

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

LocationMoveScheduleBatch Batch Design

The LocationMoveScheduleBatch program schedules location moves between zones in a zone group.

Usage

The following command runs the LocationMoveScheduleBatch job:

```
LocationMoveScheduleBatch userAlias
```

Detail

The batch finds and processes all approved location moves.

Verify the following hard-stop conditions before processing each location move:

- The move date cannot be less than today plus Location Move Lead Time.
- The currency for the old zone and new zone must be the same.
- Zone level promotions cannot overlap, unless allowed by the current system option settings.
- There can be no existing location move request in scheduled state for the same location.
- Worksheet review periods cannot overlap, unless allowed by the current system option settings.

Update the RPM_FUTURE_RETAIL table to reflect the location move:

- Price events schedule for item/locations affected by the move at the old zone level are removed from RPM_FUTURE_RETAIL.
- Conflict checking is run on RPM_FUTURE_RETAIL after events from the old zone are removed. If conflicts are encountered during conflict checking, the location move will fail and no further processing will occur for this location move.

Update the status of the location move to Scheduled.

Resolve overlapping promotions according to the current system option settings:

- Overlapping promotions are updated with appropriate exceptions/exclusions according to the current system option settings for overlapping promotions.
- Conflict checking is run on RPM_FUTURE_RETAIL to approve the updates to overlapping promotions. If conflicts are encountered during conflict checking, the location remains in Sheduled status, but the promotion exceptions/exclusions that failed conflict checking are recorded in RPM_LOC_MOVE_PROMO_COMP_DTL_EX.

Assumptions and Scheduling Notes

LocationMoveScheduleBatch must be run before LocationMoveBatch.

Primary (RPM) Tables Involved

- RPM_FUTURE_RETAIL
- RPM_LOCATION_MOVE

Threading

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

MerchExtractKickOffBatch Batch Design

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 userAlias <mode>
```

where userid is the user ID. 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

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

- Delete worksheets that are at the end of their review period.
- Get list of all strategies that need to be processed today. Create copies of the strategies as needed.
- Determine what strategies need to be grouped together based on the RPM_DEPT_AGGREGATION. WORKSHEET_LEVEL.
- 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. This allows the program to access large tables in an efficient as possible manner.

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

- Start threads based on the values in RPM_BATCH_CONTRL for MerchExtractKickOffBatch.java.
- Call RPM_EXT_SQL, a PL/SQL package, to extract RPM information. The package is called at the strategy and RPM_DEPT_AGGREGATION. WORKSHEET_LEVEL. level. It pulls large amounts of data from various RMS tables and populates the RPM_WORKSHEET_ITEM_DATA, RPM_WORKSHEET_ITEM_LOC_DATA, and RPM_WORKSHEET_ZONE_DATA tables. 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 the RPM_WORKSHEET_ITEM_DATA, RPM_WORKSHEET_ITEM_LOC_DATA, and RPM_WORKSHEET_ZONE_DATA tables.
- For each worksheet detail record created, perform the following:
  - Use the price strategy to propose a retail value.
  - Apply candidate rules.
  - Apply price guides.

The following are potential reasons why item/locations are not included in a worksheet:

- The item/location falls under an exclusion type candidate rule.
- The item/location does not have a cost on the RMS FUTURE_COST table.
- The item's market basket codes vary across locations in a zone.
- The item's link code varies across locations in a zone.

- If a link code is identified on an item/location, and there is any item within that link code (at that location) that has not been brought into the worksheet, all of the item/locations with that link code are excluded from the worksheet.
- The item's selling unit of measure varies across locations in a zone.
- The item is part of an area differential item exclusion.
- Item/locations in a single link code have varying selling unit of measures.

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

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

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

Assumptions and Scheduling Notes

The following programs must run before PriceStrategyCalendarBatch:

- PriceStrategyCalendarBatch
- LocationMoveBatch

Primary (RPM) Tables Involved

- RPM_WORKSHEET_STATUS
- RPM_WORKSHEET_ITEM_DATA
- RPM_WORKSHEET_ITEM_LOC_DATA
- RPM_WORKSHEET_ZONE_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

The NewItemLocBatch program ranges item locations by putting them into the future retail table.Item and location are fed to this program through the RPM_STAGE_ITEM_LOC table, which is populated by an RMS process.

Usage

The following command runs the NewItemLocBatch job:

```
NewItemLocBatch userAlias <status> <logical commit count>]
```

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.

To use the batch, follow these steps.

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. Run the batch (NewItemLocBatch.sh) one time to process any records remaining in RPM_STAGE_ITEM_LOC.

Detail

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

This batch also creates records on the RPM_ITEM_LOC table for all transaction level items and each location that they are ranged to as these item/locations come into the RPM system.

The batch initially create sall timelines at the transaction item and location level and applies all future/current price events that the item/location should inherit. Then the batch executes logic to generate timelines at levels above the item/location. Following that, the batch executes logic that attempts to roll up lower level timelines to the higher levels generated.

If an item/location timeline is being added under an existing higher level timeline (for example, a new location is ranged to a parent item, and the location is part of the primary zone group for that parent item), the batch process generates only the transaction item and location level timelines for all children under the parent and location that was added. The batch will not execute any logic that generates higher level timelines, nor will it attempt to roll up any lower level timelines for this data.

Assumptions and Scheduling Notes

This batch may be run ad-hoc. However, it should be noted that the item/locations to be processed 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

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

Bulk Conflict Checking

This program now utilizes the new bulk conflict checking framework inside of RPM. Each thread that is spawned by the RPM batch threading framework (threaded by item/loc) may spawn other threads (by pricing events) during its processing. See Chapter 4, "Conflict Checking," for more details.

Processing Stage Rows in Error Status

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

PriceChangeAreaDifferentialBatch Batch Design

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 userAlias
```

Additional Notes

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

Details

- 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

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

Usage

The following command runs the PriceChangeAutoApproveResultsPurgeBatch job:

```
PriceChangeAutoApproveResultsPurgeBatch userAlias
```

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

The PriceChangeAutoApproveResultsPurgeBatch program is not threaded.

PriceChangePurgeBatch Batch Design

The PriceChangePurgeBatch program deletes past price changes.

Usage

The following command runs the PriceChangePurgeBatch job:

```
PriceChangePurgeBatch userAlias
```

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

The PricechangePurgeBatch program is not threaded.

PriceChangePurgeWorkspaceBatch Batch Design

The PriceChangePurgeWorkspaceBatch program deletes abandoned price change workspace records.

Usage

The following command runs the PriceChangePurgeWorkspaceBatch job:

```
PriceChangePurgeWorkspaceBatch userAlias
```

Where the argument is the user ID.

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

The PriceChangePurgeWorkspaceBatch program is not threaded.

Price Event Execution Batch Processes

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

Executing price events require running three batch programs:

- 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 remains 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 are in "executed" status and the item-locations that failed to process remains staged in RPM_EVENT_ITEMLOC. These item-locations is 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 is 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 remains in RPM_EVENT_ITEMLOC_DEALS and the next-day batch run is guaranteed to pick these up.

Usage

The following commands need to be executed in order:

```
PriceEventExecutionBatch userAlias [restartInd Y|N]
PriceEventExecutionRMSBatch userAlias
PriceEventExecutionDealsBatch userAlias
```

Where the last argument of the PriceEventExecutionBatch indicates if the run should start over (use a value of N) or pick up where the previous run left off (use a value of Y).

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 respects 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, RPM_PROMO_COMP, RPM_PROMO_DTL, RPM_PROMO_DTL_LIST_GRP, RPM_PROMO_DTL_LIST, RPM_PROMO_DTL_MERCH_NODE, RPM_PROMO_DTL_DISC_LADDER and RPM_PROMO_ZONE_LOCATION.

RMS Interface Point

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

Threading

Two of the three batch programs involved in price event execution utilize concurrent processing. They are PriceEventExecutionBatch and PriceEventExecutionRMSBatch. Each program has a separate strategy, as the following describes.

- PriceEventExecutionBatch is threaded by a variable number of item-locations affected by the pricing events to be executed.
- To configure threading for this program, a record containing the following values must be inserted in the RPM_BATCH_CONTROL table.

Variable Name	Description
BATCH_ CONTROL_ID	System-generated sequence number.
PROGRAM_NAME	com.retek.rpm.batch.PriceEventExecutionBatch
NUM_THREADS	Number of threads to run concurrently.
THREAD_LUW_ COUNT	Number of item-locations to be executed in one thread.

Example:

For a promotions price event, where there are 50 promotion details, affecting 100,000 item-locations, in approved status--and where start and end dates are VDATE+1, the initial RPM_BATCH_CONTROL table settings are as follows:

Variable Name	Description
BATCH_ CONTROL_ID	1
PROGRAM_NAME	com.retek.rpm.batch.PriceEventExecutionBatch
NUM_THREADS	3
THREAD_LUW_ COUNT	10,000

When the PriceEventExecutionBatch program runs on VDATE, the 100,000 item-locations are grouped into ten sets of 10,000 (based on THREAD_LUW_ COUNT). A unique thread number is assigned to each group.

Each set of 10,000 item-locations details is processed in three threads, based on NUM_ THREADS. The three threads process concurrently for a group, and they may not finish at the same time. So as execution of one thread within a group completes, processing of the first thread of the next group begins--until all promotion details are processed.

- PriceEventExecutionRMSBatch is threaded by a variable number of item-locations affected by the pricing events to be executed. To configure threading for this program, a record containing the following values must be inserted in the RPM_ BATCH_CONTROL table:

Variable Name	Description
BATCH_ CONTROL_ID	System-generated sequence number.
PROGRAM_NAME	com.retek.rpm.batch.PriceEventExecutionBatchRMSBatch
NUM_THREADS	Number of threads to run concurrently.
THREAD_LUW_ COUNT	Number of item-locations to be executed in one thread.

Example:

Where there are various price events affecting 100,000 item-locations, the initial RPM_BATCH_CONTROL table settings are follows:

Variable Name	Description
BATCH_CONTROL_ID	2
PROGRAM_NAME	com.retek.rpm.batch.PriceEventExecutionRMSBatch
NUM_THREADS	3
THREAD_LUW_COUNT	10,000

When the PriceEventExecutionRMSBatch program runs on VDATE, the 100,000 item-locations are grouped into 10 sets of 10,000 item-locations (based on THREAD_LUW_COUNT). A unique thread number is assigned to each group.

Each set of 10,000 item-locations is procesesd in three threads, based on NUM_THREADS. The three threads process concurrently for a group, and they may not finish at the same time. So as execution of one thread within a group completes, processing of the first thread of the next group begins--until all item-locations are processed.

The item-locations have been staged by the PriceEventExecutionBatch program. When the PriceEventExecutionRMSBatch job is completed, these item-locations are updated in RMS.

PriceStrategyCalendarBatch Batch Design

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

Usage

The following command runs the PriceStrategyCalendarBatch job:

```
PriceStrategyCalendarBatch userAlias
```

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

The PriceStrategyCalendarBatch program is not threaded.

primaryZoneModificationsBatch Design

This batch update sthe future retail tables whenever the primary zone group definition for a merchandise hierarchy is updated/added/removed so that the future retail tables correctly reflect timelines at the zone level.

Usage

The following command runs the primaryZoneModificationsBatch job:

```
primaryZoneModificationsBatch database-connect-string LogPath ErrorPath
```

Where the first argument is the database connection string, and the second argument specifies the path from where the log file should be generated. The third argument specifies the path from where the error file should be generated.

Details

The batch calls the package RPM_PRIMARY_ZONE_UPDATE_SQL.EXECUTE. This package updates all future retail data for any merchandise hierarchies, where the primary zone group definition was modified/created/deleted and there is no other higher level merchandise hierarchy primary zone group definition that matches the new value within the system. The batch allows for future retail timelines at the zone level to accurately reflect the correct zone, and for timelines at the location level (where the location is part of the primary zone group for the given merchandise hierarchy) to correctly reference the correct zone.

Assumptions and Scheduling Notes

This process must be executed during the batch window. As it runs, other processes must not access the future retail tables.

Primary Tables Involved

- RPM_FUTURE_RETAIL
- RPM_PROMO_ITEM_LOC_EXPL
- RPM_CUST_SEGMENT_PROMO_FR
- RPM_PRIM_ZONE_MODIFICATIONS

Threading

The primaryZoneModificationsBatch program is not threaded.

ProcessPendingChunkBatch Batch Design

The ProcessPendingChunkBatch program attempts to reprocess push-back logic for threads that encountered errors.

Usage

The following command runs the ProcessPendingChunkBatch job:

```
ProcessPendingChunkBatch userAlias
```

Detail

The batch looks for any push-back threads that completed with error and any price events that encountered errors in the post-push-back logic and attempts to reprocess them using the same logic that is used during the regular conflict checking processing.

Assumptions and Scheduling Notes

The ProcessPendingChunkBatch process can be run ad-hoc - the event of a price event moving to a pending status triggering the need to run this batch. Prior to running this batch, a DBA needs to verify what issues were encountered that caused a price event to be moved to a pending status (issues like unable to extend table space). These issues need to be rectified prior to running this batch. If no action is taken to resolve these issues, the batch will likely encounter the same issues and the price events will be left in a pending status.

Primary Tables Involved

- RPM_BULK_CC_PE_CHUNK
- RPM_BULK_CC_PE_THREAD
- RPM_FUTURE_RETAIL_WS
- RPM_CLEARANCE_WS
- RPM_PROMO_ITEM_LOC_EXPL_WS
- RPM_CUST_SEGMENT_PROMO_FR_WS
- RPM_FUTURE_RETAIL
- RPM_CLEARANCE
- RPM_PROMO_ITEM_LOC_EXPL
- RPM_CUST_SEGMENT_PROMO_FR

Threading

The ProcessPendingChunkBatch program is threaded in that it will reuse the same threading logic used by the conflict checking engine when attempting to reprocess push-back threads.

PromotionArchiveBatch Batch Design

The PromotionArchiveBatch program moves promotions from active/working tables to history promotion tables.

Usage

The following command runs the PromotionArchiveBatch job:

```
PromotionArchiveBatch userAlias
```

Detail

The batch will move promotions as a whole and once the rpm_promo.end_date value is <= vdate - 1 from the active/working tables to the history versions of each table.

Assumptions and Scheduling Notes

None.

Primary Tables Involved

- RPM_PROMO
- RPM_PROMO_COMP
- RPM_PROMO_DTL
- RPM_PROMO_ZONE_LOCATION
- RPM_PROMO_DTL_LIST_GRP
- RPM_PROMO_DTL_LIST
- RPM_PROMO_DTL_DISC_LADDER
- RPM_PROMO_DTL_MERCH_NODE
- RPM_PENDING_DEAL_DETAIL
- RPM_PROMO_DEAL_LINK
- RPM_PROMO_HIST
- RPM_PROMO_COMP_HIST
- RPM_PROMO_DTL_HIST
- RPM_PROMO_ZONE_LOCATION_HIST
- RPM_PROMO_DTL_LIST_GRP_HIST
- RPM_PROMO_DTL_LIST_HIST
- RPM_PROMO_DTL_DISC_LDR_HIST
- RPM_PROMO_DTL_MERCH_NODE_HIST
- RPM_PROMO_DTL_SKULIST
- RPM_PROMO_DTL_SKULIST_HIST

Threading

The PromotionArchiveBatch program is not threaded.

PromotionPriceChangePublishBatch batch design

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 userAlias
```

The following command runs the promotionPriceChangePublishExport job:

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

Where the first argument is the database connect alias (/@db_connect_alias) and the second argument is the path where the file should be written. The path is optional; if not supplied, the path ../output is 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 the following) 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_ITEM_PAYLOAD
- RPM_PROMO_LOCATION_PAYLOAD
- RPM_PROMO_DISC_LDR_PAYLOAD
- RPM_PROMO_DTL_LIST_PAYLOAD
- RPM_PROMO_DTL_LIST_GRP_PAYLOAD
- RPM_PROMO_DTL_PAYLOAD
- RPM_PROMO_ITEM_LOC_SR_PAYLOAD
- RPM_PROMO_CREDIT_DTL
- RPM_FINANCIALS_DTL

Output File Record Types

- FHEAD (required): File identification, one line per file.
- TMBPE (optional): Event Type.
- TPDTL (required): Promotion Component Detail.
- TLLST (required): Promotion Location List (one or more per TPDTL).
- TPGRP (required): Promotion Group (one or more per TPDTL).
- TGLST (required): Promotion List (one or more per TPGRP).
- TLITM (required): Promotion Item (one or more per TGLST).
- TPDSC (required): Promotion Discount (one or more per TGLST).
- TPILSR (optional): Promotion Item Location Selling retail (one or more per TPDTL).
- TPCDT (optional): Promotion Credit Detail (one or more per TPDTL).
- TTAIL (required): Transaction tail (one per promotion).
- FPDEL (optional): 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 Identifier
	File Type	Char(5)	PROMO	Promotions
	Export timestamp	Timestamp		System clock timestamp (YYYYMMDDHHMISS)
	Format Version	Char(5)	1.0	File Format Version
TMBPE	Record Descriptor	Char(5)	TMBPE	Promotion (transaction head)
	Line id	Number(10)		Unique line identifier
	Event Type	Char(3)		CRE = Create, MOD = Modify
TPDTL	Record Descriptor	Char(5)	TPDTL	Promotion Detail Component
	Line id	Number(10)		Unique line identifier
	Promo Id	Number(10)		Promotion identifier
	Promo Comp Id	Number(10)		Promotion Component Id
	Promo Name	Char(160)		Promotion Header Name
	Promo Desc	Char(640)		Promotion Header Description
	Promo Comp Desc	Char(160)		Promotion Component Name
	Promo Type	Number(2)		Promotion Component Type

Record Name	Field Name	Field Type	Default Value	Description
	Promo Comp Detail Id	Number(10)		Promotion Component Detail identifier
	Start Date	Date		Start Date of Promotion Component Detail (YYYYMMDD)
	End Date	Date		End Date of Promotion Component Detail (YYYYMMDD)
	Apply Order	Number(1)		Application Order of the Promotion
	Threshold Id	Number(6)		Threshold identifier
	Customer Type Id	Number(10)		Customer Type identifier
TLLST	Record Descriptor	Char(5)	TLLST	Promotion Detail Component
	Line id	Number(10)		Unique line identifier
	Location Id	Number(10)		Org Node (Store or Warehouse) identifier
	Location Type	Char(1)		Org Node Type (Store or Warehouse)
TPGRP	Record Descriptor	Char(5)	TPGRP	Promotion Detail Group
	Line id	Number(10)		Unique line identifier
	Group Id	Number(10)		Group Number
TGLIST	Record Descriptor	Char(5)	TGLIST	Promotion Group List
	Line id	Number(10)		Unique line identifier
	List Id	Number(10)		List identifier
	Reward Application	Number(1)		How this reward is applied to the promotion detail.
	Description	Char(120)		Description
TLITM	Record Descriptor	Char(5)	TLITM	Promotion Group List
	Line id	Number(10)		Unique line identifier
	Item Id	Char(25)		Transaction Item Identifier
TPDSC	Record Descriptor	Char(5)	TPDSC	Discount Detail for List
	Line id	Number(10)		Unique line identifier
	Change Type	Number(2)		Change Type
	Change Amount	Number(20,4)		Change Amount
	Change Currency	Char(3)		Change Currency
	Change Percent	Number(20,4)		Change Percent
	Change Selling UOM	Char(4)		Change Selling UOM
	Qual Type	Number(2)		Qualification Type
	Qual Value	Number(2)		Qualification Value

Record Name	Field Name	Field Type	Default Value	Description
	Change Duration	Number(20,4)		Change Duration
TPILSR	Record Descriptor	Char(5)	TPILSR	Items in Promotion
	Line id	Number(10)		Unique line identifier
	Item Id	Char(25)	TTAIL	Transaction Item Identifier
	Selling Retail	Number(20,4)		Selling retail of the item
	Selling UOM	Char(4)		Selling UOM of the item
	Location Id	Number(10)		Org Node (Store or Warehouse) identifier
TPCDT	Record Descriptor	Char(5)	TPCDT	Credit Detail
	Credit Detail Id	Number(10)		Credit Detail Id
	Line Id	Number(10)		Unique line id
	Credit Type	Char(40)		Credit Type
	binNumber From	Number(10)		Bin Number From
	binNumberTo	Number(10)		Bin Number To
	Commission Rate	Number(10)		Commission Rate
	Comments	Char(160)		Comments
TTAIL	Record Descriptor	Char(5)	TTAIL	Transaction Tail
	Line id	Number(10)		Unique line identifier
FPDEL	Record Descriptor	Char(5)	FPDEL	Delete Promotion
	Line id	Number(10)		Unique line identifier
	Promo ID	Number(10)		The ID of the promotion
	Promo Comp Id	Number(10)		Promotion Component Id
	Promo Comp Detail Id	Number(10)		Promotion Component Detail identifier
	Group Id	Number(10)		Group Number
	List Id	Number(10)		List identifier
	Item Id	Char(25)		Transaction Item Identifier for item
	Location Id	Number(10)		Org Node (Store or Warehouse) identifier
FTAIL	Record Descriptor	Char(5)	FTAIL	File tail marker
	Line id	Number(10)		Unique line identifier
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

Assumptions and Scheduling Notes

PromotionPriceChangePublishBatch should be run after the WorksheetAutoApproveBatch.

PromotionPriceChangePublishExport should be executed as follows:

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

Threading

The PromotionPriceChangePublish program is threaded. The LUW is a single rpm_price_event_payload record. Multi-buy promotions are not 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

The PromotionPurgeBatch program deletes old and rejected promotions.

Usage

The following command runs the PromotionPurgeBatch job:

```
PromotionPurgeBatch userAlias
```

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_LOC_MOVE_PROMO_COMP_DTL_EX
- RPM_CONFLICT_CHECK_RESULT
- RPM_PROM_COMP_AGG_TBL
- RPM_PROMO_DEAL_LINK
- RPM_PENDING_DEAL_DETAIL
- RPM_PENDING_DEAL
- RPM_PROMO_DTL_MERCH_NODE_HIST
- RPM_PROMO_DTL_DISC_LDR_HIST
- RPM_PROMO_DTL_LIST_HIST
- RPM_PROMO_DTL_LIST_GRP_HIST
- RPM_PROMO_ZONE_LOCATION_HIST
- RPM_PROMO_DTL_HIST
- RPM_PROMO_COMP_HIST
- RPM_PROMO_HIST
- RPM_PROMO_CREDIT_DTL

Threading

This program is threaded.

PurgeBulkConflictCheckArtifacts Batch Design

The purgeBulkConflictCheckArtifacts program cleans up the working tables used by Bulk Conflict Checking and Chunk Conflict Checking engines.

Usage

The following command runs the purgeBulkConflictCheckArtifacts job:

```
purgeBulkConflictCheckArtifacts userAlias
```

Where the argument is the user ID.

Detail

The current release of RPM comes with the Bulk Conflict Checking and Chunk Conflict Checking engines. These engines use several working tables to do their processing. In normal conditions, these tables are supposed to be deleted at the end of the Bulk Conflict Checking process. But if there are any environment issues, it is possible that there will be some records left in these tables. This batch program cleans up those working tables. This batch makes sure that the system has a clean set of working tables for Bulk Conflict Checking and Chunk Conflict Checking for the next day. Users using the application with too many records left in these working tables could impact the performance of both Bulk Conflict Checking and Chunk 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
- RPM_FUTURE_RETAIL_WS
- RPM_PROMO_ITEM_LOC_EXPL_WS
- RPM_CUST_SEGMENT_PROMO_FR_WS
- RPM_CLEARANCE_WS

PurgeExpiredExecutedOrApprovedClearancesBatch Batch Design

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

Usage

The following command runs the PurgeExpiredExecutedOrApprovedClearancesBatch job:

```
PurgeExpiredExecutedOrApprovedClearancesBatch <database-connect-alias> <log path>
<error path>
```

Where the first argument is the RPM database-connect-alias (/@db_connection_alias). 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 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
- RPM_CLEARANCE_RESET

Threading

The PurgeExpiredExecutedOrApprovedClearancesBatch program is not threaded.

PurgeLocationMovesBatch Batch Design

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

Usage

The following command runs the PurgeLocationMovesBatch job:

```
PurgeLocationMovesBatch userAlias
```

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

The PurgeLocationMovesBatch program is not threaded.

PurgePayloadsBatch Batch Design

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

Usage

The following command runs the PurgePayloadsBatch job:

```
purgePayloadsBatch.shsh db_connection_alias <publish-status>_<log-path>
```

Where the first argument is the database connection alias(/@db_connection_alias).

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:

Detail

Table 7–1

<publish-status>	Description	
1	This setting results in purged price events that have been exported by RMPtoORPOSPublishExport.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 PurgePayloadsBatch program purges the price events 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:

Table 7–2

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_FIN_DTL_PAYLOAD
- RPM_PRICE_EVENT_PAYLOAD

Threading

The PurgePayloadsBatch program is not threaded.

PurgeUnusedAndAbandonedClearancesBatch Batch Design

The PurgeUnusedAndAbandonedClearancesBatch program deletes unused and rejected clearances.

Usage

The following command runs the PurgeUnusedAndAbandonedClearancesBatch job:

```
PurgeUnusedAndAbandonedClearancesBatch <database-connect-alias> <log path> <error path>
```

Where the first argument is the RPM database-connect-alias (/@db_connection_alias). 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 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
- RPM_CLEARANCE_RESET

Threading

The PurgeUnusedAndAbandonedClearancesBatch program is not threaded.

RefreshPosDataBatch Batch Design

The RefreshPosDataBatch batch is intended to provide the Point of Sale (POS) application with data refresh capability. This batch generates all future pending pricing information for a store, starting with a specified date that is passed as a parameter.

It then looks for pending price events pertaining to the specified store ID in the RPM_ FUTURE_RETAIL table and populates the payload tables with pending price event information, starting from the specified action date.

The information is then sent to POS for the purpose of refreshing all future price events for a store. The batch is not intended to be used for a new store or a new instance of POS in an existing store.

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 program deletes the contents of the payload tables listed as follows. 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 data 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_PRICE_EVENT_PAYLOAD
- RPM_PRICE_CHG_PAYLOAD
- RPM_CLEARANCE_PAYLOAD
- RPM_PROMO_DTL_PAYLOAD
- RPM_PROMO_DISC_LDR_PAYLOAD
- RPM_PROMO_DTL_LIST_GRP_PAYLOAD
- RPM_PROMO_DTL_LIST_PAYLOAD
- RPM_PROMO_DTL_PAYLOAD
- RPM_PROMO_FIN_DTL_PAYLOAD
- RPM_PROMO_ITEM_LOC_SR_PAYLOAD
- RPM_PROMO_ITEM_PAYLOAD
- RPM_PROMO_LOCATION_PAYLOAD

Output

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

Threading

The RefreshPosDataBatch program is threaded based on department count.

RegularPriceChangePublishBatch Batch Design

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 the RegularPriceChangePublishBatch.

Usage

The following command runs the RegularPriceChangePublishBatch job:

```
RegularPriceChangePublishBatch userAlias
```

The following command runs the regularPriceChangePublishExport job:

```
rregularPriceChangePublishExport.sh database-connect-alias path
```

Where the first argument is the database connect alias (/@db_connection_alias) and the second argument is the path where the file should be written. The path is optional; if not supplied, the path ../output is used.

Detail

The batch 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 Files

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 identifier
	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 identifier
	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
	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 of 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

Record Name	Field Name	Field Type	Default Value	Description
FDELE	Record Descriptor	Char(5)	FDELE	File Detail Delete Marker (1 per price change delete)
	Line id	Number(10)		Unique line identifier
	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 identifier
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

Assumptions and Scheduling Notes

RegularPriceChangePublishBatch should be run after the WorksheetAutoApproveBatch.

RegularPriceChangePublishExport should be run after every successful run of RegularPriceChangePublishBatch.

Primary Tables Involved

- RPM_PRICE_EVENT_PAYLOAD
- RPM_PRICE_CHG_PAYLOAD

Threading

This program 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

The RPMtoORPOSPublishBatch program formats and stages output of different price events like PriceChange, Clearance and Promotions. The RPMtoORPOSPublishExport shell script produces an xml file based on the output of the RPMtoORPOSPublishBatch.

Usage

The following command runs the RPMtoORPOSPublishBatch job:

RPMtoORPOSPublishBatch <database-connect-alias> <log path> <error path>

Where the first argument is the RPM database-connect-alias (/@db_connection_alias). 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 took data for different price events in the payload and price event tables. Each price events have their own payload tables.

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	Promotion	RPM_PROMO_DTL_PAYLOAD RPM_PRICE_EVENT_PAYLOAD RPM_PROMO_LOCATION_PAYLOAD RPM_PROMO_DTL_LIST_GRP_PAYLOAD RPM_PROMO_DTL_LIST_PAYLOAD RPM_PROMO_ITEM_PAYLOAD

The data collected from all these payload tables is formatted and inserted in different tables.

S.No.	Price Event	Related Tables
1	Price Change	RPM_ORPOS_PRICE_CHANGE_PUBLISH
2	Clearance	RPM_ORPOS_PRICE_CHANGE_PUBLISH
3	Simple Promo	RPM_ORPOS_SIMPLE_PROMO_PUBLISH
4	Threshold Promo	RPM_ORPOS_CMPLX_PROMO_PUBLISH
5	Complex Promo	RPM_ORPOS_CMPLX_PROMO_PUBLISH

Output

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

Assumptions and Scheduling Notes

RPMtoORPOSPublishBatch should be run after WorksheetAutoApproveBatch. This batch should be executed before the following batches.

- RegularPriceChangePublishBatch
- RegularPriceChangePublishExport
- ClearancePriceChangePublishBatch
- ClearancePriceChangePublishExport
- PromotionPriceChangePublishBatch
- PromotionPriceChangePublishExport

Primary Tables Involved

- RPM_ORPOS_PRICE_CHANGE_PUBLISH
- RPM_ORPOS_SIMPLE_PROMO_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

The RPMtoORPOSPublishExport program calls the SQL script, RPMtoORPOSSpoolMsg.sql. This script spools the data collected from different publish tables of different price events (Price Change, Clearance and Promotion). Spooled file contains the data in XML format. The XML file complies with the given XSD standard.

Usage

The following command runs the RPMtoORPOSPublishExport job:

```
RPRPMtoORPOSPublishExport <database-connect-alias> <Number of slots>  <log path>
<error path> <Export path> <credit promotion indicator>
```

Where the first argument is the RPM database-connect-alias (/@db_connection_alias); the second argument is the number of slots (or 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; and the sixth argument is credit promotion, where N (default) creates no credit promotion XML files and Y generates credit promotion XML files. The export path is optional. If it is not supplied, the pwd (present working directory) will be used for the same.

Note: The number of slots variable must be greater than zero. The log and error path parameters must be a valid directory at the same level as PWD.

Detail

The batch looks for data in different publish tables.

- The batch invokes a SQL script by passing the store_id. In the SQL script, the spool filename is generated in the following way: PricingExtract_<store_id>.xml. A file is generated for all the stores whether price events exist or not. Each price event possesses different structures. These structures and values comply with the XSD standard.
- The batch invokes a SQL script for generating the credit promotions by passing the store_id. In the SQL script, the spool file name is generated in the following way: CreditPromotion_<store_id>.xml.
- The batch script also invokes a new spool routine to compile item import XML with list group IDs. The XML, ItemClassificationMod.xml, contains mod item messages that include the items and all applicable list group IDs for all stores.

OutputFile

The output file is an XML file (PricingExtract_<store_id>.xml) that is generated on a per store basis. The PricingExtract_<store_id>.xml file should comply with the XSD standard and contain three different types of tag structures. The three types of tag structures are as follows:

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/Multi-buy Promotion	Discount Rule
4	Credit Promotion	Credit Promotion

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_PROMO_PUBLISH
- RPM_ORPOS_CREDIT_PROMO_PUBLISH
- RPM_ORPOS_CMPLX_PROMO_PUBLISH

Threading The RPMtoORPOSPublishExport program 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
```

> **Note:** When the RPM application server is installed on a SunOS operating system, the shell interpreter specification for this batch needs to be manually modified at the time of installation. It should be updated to use the Korn Shell interpreter rather than the Bourne Shell interpreter.

statusPageCommandLineApplication Batch Design

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. The checks are done with SQL counts; each check should return zero 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 RPM_FUTURE_ RETAIL table per item, location, and action date.

The command usage is as follows:

```
statusPageCommandLineApplication.sh userAlias [phase-choice]
 [max-rows-choice]
```

Valid values for phase choice are as follows:

Choice	Description
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 retailUser S
```

The following 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_
76208b17_Tie.executeCommand__com_retek_platform_service_ServiceCommand(_
EJSRemoteStatelessCommandExecutionService_76208b17_Tie.java(Compiled Code))
at com.retek.platform.service.impl._EJSRemoteStatelessCommandExecutionService_
76208b17_Tie._invoke(_EJSRemoteStatelessCommandExecutionService_76208b17_
Tie.java(Compiled 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
MULTIBUYPROMOCRE is ON **********The above exception indicates that
we have passed **************************
****************************************
Starting Report
com.retek.rpm.statuspage.RpmRibMessageCheck Passed PRMPRCCHG.PRMPRCCHGMOD
MULTIBUYPROMOMOD is ON **********The above exception indicates that
we have passed **************************
****************************************
Starting Report
com.retek.rpm.statuspage.RpmRibMessageCheck Passed PRMPRCCHG.PRMPRCCHGDEL
MULTIBUYPROMODEL is ON **********The above exception indicates that
we have passed **************************
****************************************
Starting Report
RpmJmsServerCheck Passed

Done.
```

TaskPurgeBatch Batch Design

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

Usage

The following command runs the TaskPurgeBatch job:

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

Where the second argument is number if purge days (positive integer) and the last argument is the complete status indicator which is either 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 is taken with respect to the VDATE in PERIOD table. All the entries before VDATE – purgeDays are purged. Also the status indicator is considered while the values are being purged. If the status indicator is Y, then only tasks with completed status are 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
- RPM_CHUNK_CC_TASK

Threading

TaskPurgeBatch.java is not threaded.

WorksheetAutoApproveBatch Batch Design

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 userAlias
```

Where the argument is the user ID.

Detail

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

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

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

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

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

If dynamic area differentials are being used in the system, any secondary area worksheet detail records that exist are also processed. If the secondary area is marked as auto approve, the secondary worksheet detail record goes through the same logic as the original worksheet detail (depending on its state). If the secondary area is not marked as auto approve, the secondary worksheet detail record has a retail proposed for it and moves into new status and becomes 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_ITEM_DATA
- RPM_WORKSHEET_ITEM_LOC_DATA
- RPM_WORKSHEET_ZONE_DATA
- RPM_PRICE_CHANGE
- RPM_CLEARANCE
- RPM_FUTURE_RETAIL

- RPM_AREA_DIFF_PRIM
- RPM_AREA_DIFF
- RPM_MAINT_MARGIN_ERR
- RPM_MAINT_MARGIN_ERR_DTL

Threading

This program is threaded but does not use the RPM_BATCH_CONTROL table. Please see Chapter 4, "Conflict Checking," for thread configuration information.

ZoneFutureRetailPurgeBatch Batch Design

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

Usage

The following command runs the PriceChangeAutoApproveResultsPurgeBatch job:

```
ZoneFutureRetailPurgeBatch userAlias
```

Where the argument is the user ID .

Detail

The ZoneFutureRetailPurgeBatch program deletes all zone/item price change actions which:

- Have an ACTION_DATE value prior to VDATE.
- Have been superseded by at least one other change action whose ACTION_DATE is also prior to VDATE.

The effect is that, 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

This program is not threaded.