

# Retek® Price Management™ 11.0.4

## Operations Guide



---

**Corporate Headquarters:**

Retek Inc.  
Retek on the Mall  
950 Nicollet Mall  
Minneapolis, MN 55403  
USA  
888.61.RETEK (toll free US)  
Switchboard:  
+1 612 587 5000  
Fax:  
+1 612 587 5100

**European Headquarters:**

Retek  
110 Wigmore Street  
London  
W1U 3RW  
United Kingdom  
Switchboard:  
+44 (0)20 7563 4600  
Sales Enquiries:  
+44 (0)20 7563 46 46  
Fax:  
+44 (0)20 7563 46 10

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

No part of this documentation may be reproduced or transmitted in any form or by any means without the express written permission of Retek Inc., Retek on the Mall, 950 Nicollet Mall, Minneapolis, MN 55403, and the copyright notice may not be removed without the consent of Retek Inc.

Information in this documentation is subject to change without notice.

Retek provides product documentation in a read-only-format to ensure content integrity. Retek Customer Support cannot support documentation that has been changed without Retek authorization.

The functionality described herein applies to this version, as reflected on the title page of this document, and to no other versions of software, including without limitation subsequent releases of the same software component. The functionality described herein will change from time to time with the release of new versions of software and Retek reserves the right to make such modifications at its absolute discretion.

Retek® Price Management™ is a trademark of Retek Inc.

Retek and the Retek logo are registered trademarks of Retek Inc.

This unpublished work is protected by confidentiality agreement, and by trade secret, copyright, and other laws. In the event of publication, the following notice shall apply:

©2005 Retek Inc. All rights reserved.

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

Printed in the United States of America.

## Customer Support

### Customer Support hours

Customer Support is available 7x24x365 via email, phone, and Web access.

Depending on the Support option chosen by a particular client (Standard, Plus, or Premium), the times that certain services are delivered may be restricted. Severity 1 (Critical) issues are addressed on a 7x24 basis and receive continuous attention until resolved, for all clients on active maintenance. Retek customers on active maintenance agreements may contact a global Customer Support representative in accordance with contract terms in one of the following ways.

Contact Method	Contact Information
----------------	---------------------

E-mail	support@retек.com
--------	-------------------

Internet (ROCS)	<a href="https://rocs.retek.com">rocs.retek.com</a> Retek's secure client Web site to update and view issues
-----------------	---

Phone	+1 612 587 5800
-------	-----------------

Toll free alternatives are also available in various regions of the world:

Australia	+1 800 555 923 (AU-Telstra) or +1 800 000 562 (AU-Optus)
France	0800 90 91 66
Hong Kong	800 96 4262
Korea	00 308 13 1342
United Kingdom	0800 917 2863
United States	+1 800 61 RETEK or 800 617 3835

Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403
------	---

### When contacting Customer Support, please provide:

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

# Contents

<b>Chapter 1 – Introduction .....</b>	<b>1</b>
Overview—what is RPM? .....	1
Who this guide is written for .....	1
Technical architecture overview .....	2
RPM’s integration points into the retail enterprise .....	2
Where you can find more information.....	3
<b>Chapter 2 – Backend system administration and configuration..</b>	<b>5</b>
Supported Retek products .....	5
Supported environments .....	5
Exception handling .....	5
Configuration files .....	6
rpm11.jnlp .....	6
Datasource configuration in container.....	6
rib_user.properties .....	6
Configuration for Retek Service Layer (RSL) with services_rpm.xml .....	7
RSM-related configuration file .....	7
Logging .....	8
Jakarta commons logging .....	8
Log4j.xml .....	8
Logging levels .....	8
Output files .....	9
Hibernate logging .....	9
Transaction timeout and client inactivity timeout .....	10
TaskMDB.....	10
EJBs used by TaskMDB.....	10
Tables used by TaskMDB .....	11
Disabling RIB publishing in RPM.....	11
Internationalization and localization.....	11
Translation.....	12
<b>Chapter 3 – Technical architecture .....</b>	<b>13</b>
Overview.....	13
The layered model.....	14
Client .....	15
Application services layer (stateless session beans).....	15

Core services layer .....	16
Persistence layer .....	16
Database layer .....	16
Security .....	16
Asynchronous processing .....	17
RPM-related Java terms and standards .....	18
<b>Chapter 4 – Integration functional dataflows.....</b>	<b>21</b>
A note about the merchandising system interface .....	21
Integration interface dataflow diagram .....	21
Integration interface dataflow description .....	22
From Retek Allocation to RPM.....	22
From RPM to Retek Allocation.....	22
From RPM to RMS .....	22
From RMS to RPM .....	23
From RPM to RSM .....	24
From RSM to RPM .....	24
From RPM to ReSA .....	24
From RPM to SIM/ISO and SIM/ISO to RPM .....	25
From RPM to RDW .....	26
Price management status page .....	26
<b>Chapter 5 – Methods of integration .....</b>	<b>27</b>
RPM and the Retek Integration Bus (RIB) .....	27
The XML message format.....	27
Message publication processing .....	28
Message subscription processing .....	28
Publishers mapping table.....	29
Subscribers mapping table.....	30
Functional descriptions of messages .....	30
RPM and the Retek Service Layer (RSL).....	33
Functional description of the class using RSL .....	34
Persistence layer integration .....	34
RMS tables accessed through the persistence layer .....	34
RMS packages and methods accessed through RPM’s persistence layer .....	37
RPM views based on RMS tables .....	37
RPM packages called by RMS.....	38
<b>Chapter 6 – Functional design .....</b>	<b>39</b>
Overview .....	39
Functional assumptions.....	39

Functional overviews .....	40
Zone structures .....	40
Codes .....	41
Price changes, promotions, clearances, and promotion constraint .....	42
Pricing strategies .....	44
Price inquiry .....	52
Worksheet.....	53
Calendar .....	56
Aggregation level .....	57
Location moves .....	58
Concurrency considerations.....	59
Pessimistic data locking .....	59
Pessimistic workflow locking .....	59
Last user wins.....	60
Optimistic data locking .....	60
Concurrency solution/Functional area matrix .....	60
<b>Chapter 7 – Java and RETL batch processes .....</b>	<b>63</b>
Java batch processes .....	63
Java batch process architectural overview .....	63
Running a Java-based batch process .....	63
Additional notes .....	64
Script catalog.....	64
Scheduler and the command line.....	64
Functional descriptions and dependencies .....	65
Batch process scheduling .....	65
Threading and the RPM_BATCH_CONTROL table .....	66
Return value batch standards.....	66
Return values.....	66
Batch logging .....	66
LocationMoveBatch batch design .....	67
MerchExtractKickOffBatch batch design .....	68
PriceChangeAutoApproveResultsPurgeBatch batch design .....	71
PriceChangePurgeBatch batch design.....	72
PriceChangePurgeWorkspaceBatch batch design .....	73
PriceEventExecutionBatch batch design.....	74
PriceStrategyCalendarBatch batch design.....	76
PromotionPurgeBatchbatch design .....	77
PurgeExpiredExecutedOrApprovedClearancesBatch batch design .....	78
PurgeLocationMovesBatch batch design .....	79
PurgeUnusedAndAbandonedClearancesBatch batch design .....	80
WorksheetAutoApproveBatch batch design .....	81
RETL program overview for RPM extractions .....	83
Architectural design .....	83
RPM extraction architecture.....	84
Configuration .....	84
RETL.....	84

RETL user and permissions .....	84
Environment variables.....	84
dwi_config.env settings.....	85
Program features.....	86
Program status control files .....	86
Restart and recovery .....	86
Message logging.....	87
Daily log file.....	87
Format .....	87
Program error file .....	87
Schema files .....	88
Resource files .....	88
Typical run and debugging situations.....	88
RETL extractions program list.....	89
RETL extract program flow diagrams.....	90
Legend.....	90
Program flow diagram.....	90
Application programming interface (API) flat file specifications .....	90
API format.....	90
File layout.....	91
General business rules and standards common to all APIs .....	91



# Chapter 1 – Introduction

This operations guide serves as a Retek Price Management (RPM) reference to explain ‘backend’ processes. The guide is designed so that you can view and understand key system administered functions, including batch processing, the flow of data into and out of the application, and the application’s behind-the-scenes processing of data.

## Overview—what is RPM?

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 complex buy/get 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.

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.

## Who this guide is written for

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

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

## Technical architecture overview

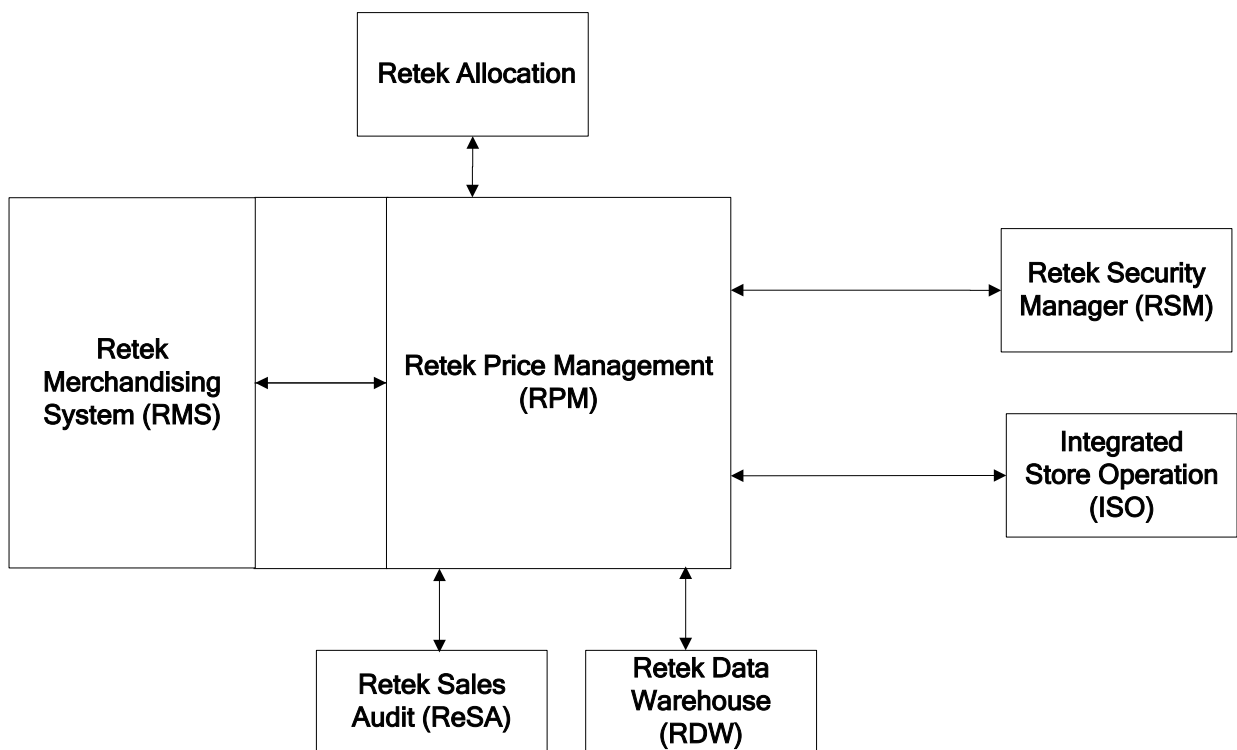
RPM's architecture is built upon a layered model. That is, layers of the application communicate with one another through an established hierarchy and are only able to communicate with neighboring layers. Any given layer need not be concerned with the internal functional tasks of any other layer.

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.

For a more detailed description of RPM's technical architecture, see "Chapter 3 – Technical architecture".

## RPM's integration points into the retail enterprise

The following high-level diagram shows the overall direction of the data among systems and products across the enterprise. For a detailed description of this diagram, see "Chapter 4 – Integration functional dataflows".



**RPM-related dataflow across the enterprise**

## Where you can find more information

You can get more information pertaining to RPM from the following sources:

- RPM front-end documentation (for example, the RPM User Guide and online help)
- RPM Installation Guide
- RPM Batch Schedule
- Retek Merchandising System (RMS) product documentation
- Retek Integration Guide and other RIB-related documentation
- Retek Security Manager (RSM) product documentation
- Applicable third-party documentation (such as for Hibernate, and so on)



# Chapter 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 Retek products

This version of RPM is compatible with the following Retek products:

- RMS 11.0.3 (including Retek Sales Audit)
- Retek Allocation 11.0.2
- RIB 11.1
- RSL 11.1
- RDW 11.0
- SIM (ISO) 11.0.0.1
- RETL 11.2.2
- Retek Navigator 11.0.1
- RSM 11.1.1

## Supported environments

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

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

## Exception handling

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

- System exceptions  
For example, server connection and/or database issues are system exceptions. System exceptions 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.

## **rpm11.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. The URL at which RPM Help is installed can also be set in this file if the default RPM Help URL is not used (see the Installation Guide for details).

## **Datasource configuration in container**

Values that can require configuration for WebSphere include those listed below. See the RPM Installation Guide for more information.

- The JDBC driver path
- The userID and password of the RMS/RPM database schema owner
- Data source name (for example, RPM)

## **rib\_user.properties**

There must be a rib\_user.properties file located in conf/retek. This properties file is used to log on to the system at the beginning of each injector. Any data changes that happen as a result of the RIB has this user listed if users are tracked with regard to create/update/approve actions. The file is populated with the values below. For more information about the RIB, see “Chapter 5 – Methods of integration” and RIB documentation.

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

## Configuration for Retek 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 – Methods of integration".

For example:

`retex/services_rpm.xml`:

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

`retex/service_flavors.xml`:

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

## RSM-related configuration file

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

# Logging

## Jakarta commons logging

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

In RPM, Log4j is the library under commons logging.

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

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

## Log4j.xml

The logging mechanism that is used for RPM is log4j.xml, which is the same as the 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>

## 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's default logging output files are shown below. They are the standard WebSphere logging output files.

For example:

- SystemOut.txt
- SystemError.txt

### 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 both a transaction timeout and a client inactivity timeout. Websphere documentation defines these timeouts as follows:

- **Transaction timeout**  
The maximum duration, in seconds, for transactions on the application server. Any transaction that is not requested to complete before this timeout will be rolled back.
- **Client inactivity timeout**  
The maximum duration, in seconds, between transactional requests from a remote client. Any period of client inactivity that exceeds this timeout results in the transaction's being rolled back in the application server.

To set up these timeouts, please follow these steps:

1. Log in to the WebSphere Application Server
2. Expand the '+ Servers' symbol
3. Click the 'Application Server' link.
4. Click the Server Name of your server application.
5. Click the 'Transaction Service' link.
6. Edit the 'Total transaction lifetime timeout' field until it reflects your desired value (for example, 600 seconds).
7. Edit the 'Client inactivity timeout' field until it reflects your desired value (for example, 600 seconds)/
8. Click 'Apply'.

## TaskMDB

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

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

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

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

## EJBs used by TaskMDB

com.retek.platform.app.taskengine.service.impl.TaskEngineServiceEjb

## Tables used by TaskMDB

- TASK  
Current, past, and pending tasks to be executed.
- TASK\_ALERT\_MODE, TASK\_STATE  
Lookup tables used by the TaskEngineServiceEjb.
- ALERTS, ALERT\_RECEIVER, ALERT\_STATUS, ALERT\_STATUS\_DSC  
Tables used for sending alerts to users about task status.

## Disabling RIB publishing in RPM

The steps below describe how a retailer can disable RIB publishing in RPM. One reason for this procedure is that a retailer may wish to run a test but *not* want results published to the RIB. For more information about the RIB, see “Chapter 5 – Methods of integration” and RIB documentation.

1. Log in to the WebSphere Admin Console.
2. Expand "Servers" from the menu.
3. Click "Application Servers" from the sub-menu.
4. Click the link that is the name of your server (for example, "perfqc").
5. Click the "Process Definition" link.
6. Click the "Java Virtual Machine" link.
7. In the "Generic JVM arguments" field, type the following value:  
`"-Dretex.no.rib=true"`

## Internationalization and localization

RPM has been internationalized to handle multiple languages in the same instance. That is, the technical infrastructure of RPM supports languages other than English.

An application that can run in various languages must be transformed into somewhat of a ‘generic’ product. That is, the features of the application that could be specific to just one language or locale (such as text, date formatting, and so on) must not be hard-coded into the software. Instead, locale-specific information is intentionally placed in files external to the application. The content of these files is interface related, as distinct from executable code.

The text in the .properties files below is translated so that the interface functions in local settings. When a country code other than the default is used, the retailer populates the \_xx.properties files below, where the retailer's 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's columns)

As shown below, the properties files can be found in rpm\_client\_properties.jar and rpm\_server\_properties.jar.

- rpm\_client\_properties.jar
  - rpm11-ui/src/com/retex/rpm/gui/Resources.properties
  - rpm11-ui/src/com/retex/rpm/gui/worksheet/Worksheet\_column\_names.properties
- rpm\_server\_properties.jar
  - rpm11-server/conf/retex/messages.properties
  - rpm11-server/conf/retex/codes.properties
  - rpm11-server/conf/retex/application\_definition\_rpm\_messages.properties

## 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, among others:

- Graphical user interface (GUI)
- Online help
- Some print documentation
- Error messages

## Chapter 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

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.

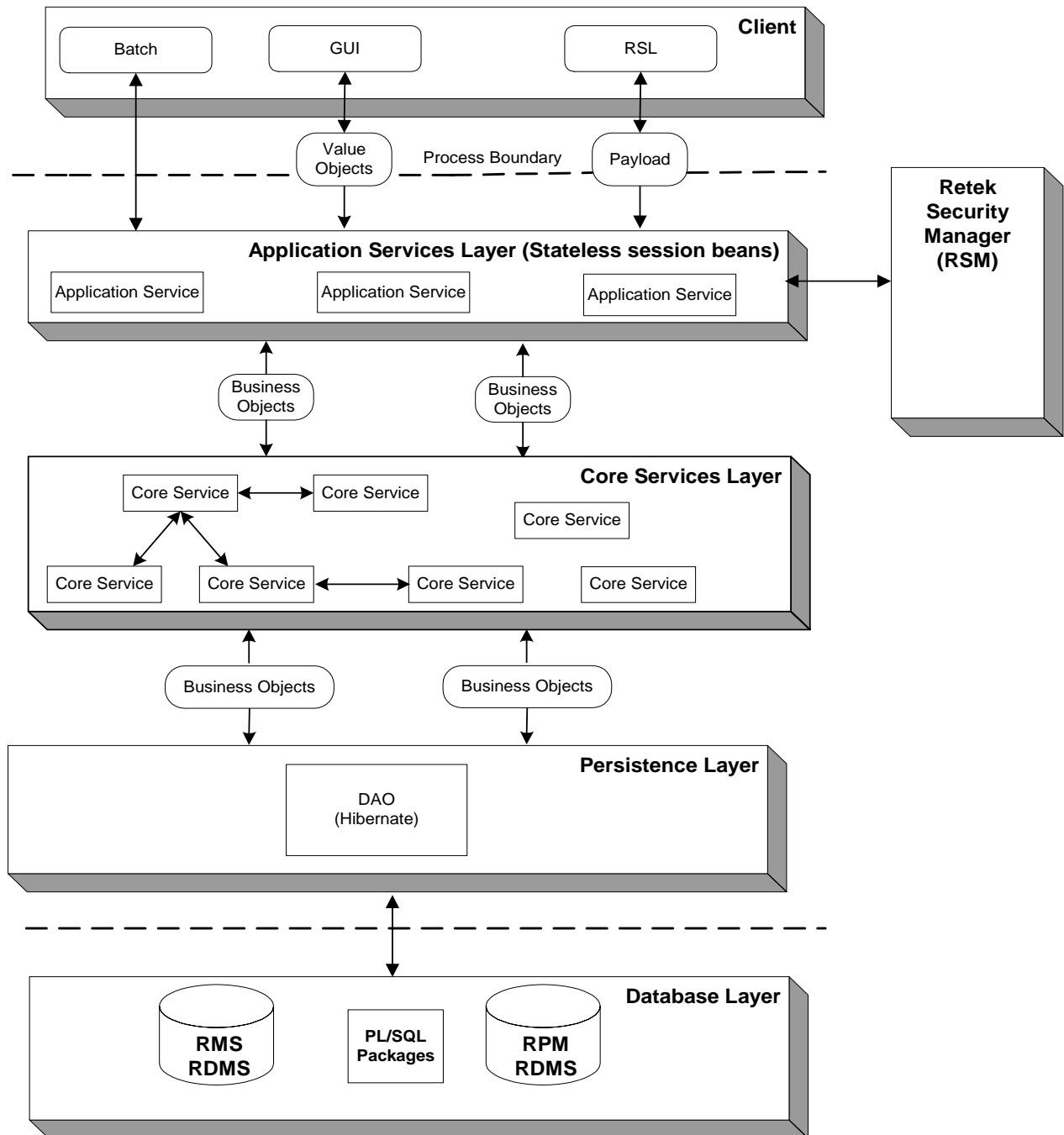
Conceptually, RPM's J2EE architecture is built upon 4-layers and implements what is defined as a service-oriented architecture. Such an architecture is essentially a collection of services that pass data, perform business processing, coordinate system activities, and render data into abstract objects. 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.



RPM's technical architecture

## 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 Retek 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 – Methods of integration". 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. (Note that 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, and so on.)

An important way that application services accept incoming data from a client is via 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 via 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 datasources, RPM and RMS.

### Security

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



## Asynchronous processing

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

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

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

- Price Changes/Clearances
- Promotions
- Worksheet



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

### Synchronous vs. asynchronous processing

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

Conflict checking is always done synchronously in the following situations:

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

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

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

Conflict checking for location moves is expected to operate on extremely large volume of data. It is not acceptable for the user to wait long periods of time for the processing of their request. Thus, conflict checking for location moves is always done asynchronously.

Constraints are performed asynchronously at the same time as the conflict check. When you execute an action that kicks off a conflict check, you indicate whether you want constraints checked. If constraints are checked, they are handled just like conflict check errors.

### Asynchronous processing flow

1. The client requests the application server to perform a process on a business object or set of business objects.
2. The server's application service layer extracts information about the request (type of business object, identifying ID, and requested action).
3. When RPM determines that a task is eligible for asynchronous processing:
  - A JMS message is published to the queue that contains specific information about the task.
  - A record is inserted into the 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 TaskMDB. TaskMDB is a message driven bean used to facilitate RPM's asynchronous processing capability. Message Driven Beans acts as listeners to specified queues for messages.
5. The task passes information to the application service layer.
6. The application service layer performs a state transition on the requested business object (for example, approving a particular price change) based on the information passed by the task.
7. The results of this transition (for example, success or failure messages, conflict details) are recorded in the RPM\_CONFLICT\_CHECK\_RESULT table.
8. The application service inserts a record into the ALERT table.
9. The client periodically polls the Alerts application service which looks for new alerts in the ALERTS table.

## RPM-related Java terms and standards

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

### ACID

ACID represents the four properties of every transaction:

- **Atomicity:** Either all of the operations bundled in the transaction are performed successfully or none of them are performed.
- **Consistency:** The transaction must leave any and all datastores that are affected by the transaction in a consistent state.
- **Isolation:** From the 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 datastore 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), version 1.4.1**

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 lowercased verb. The first letter of each internal word is capitalized.

### **Persistence**

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

### **Remote interface**

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

## Chapter 4 – Integration functional dataflows

This chapter provides a functional overview of how RPM integrates with other systems (including other Retek systems).

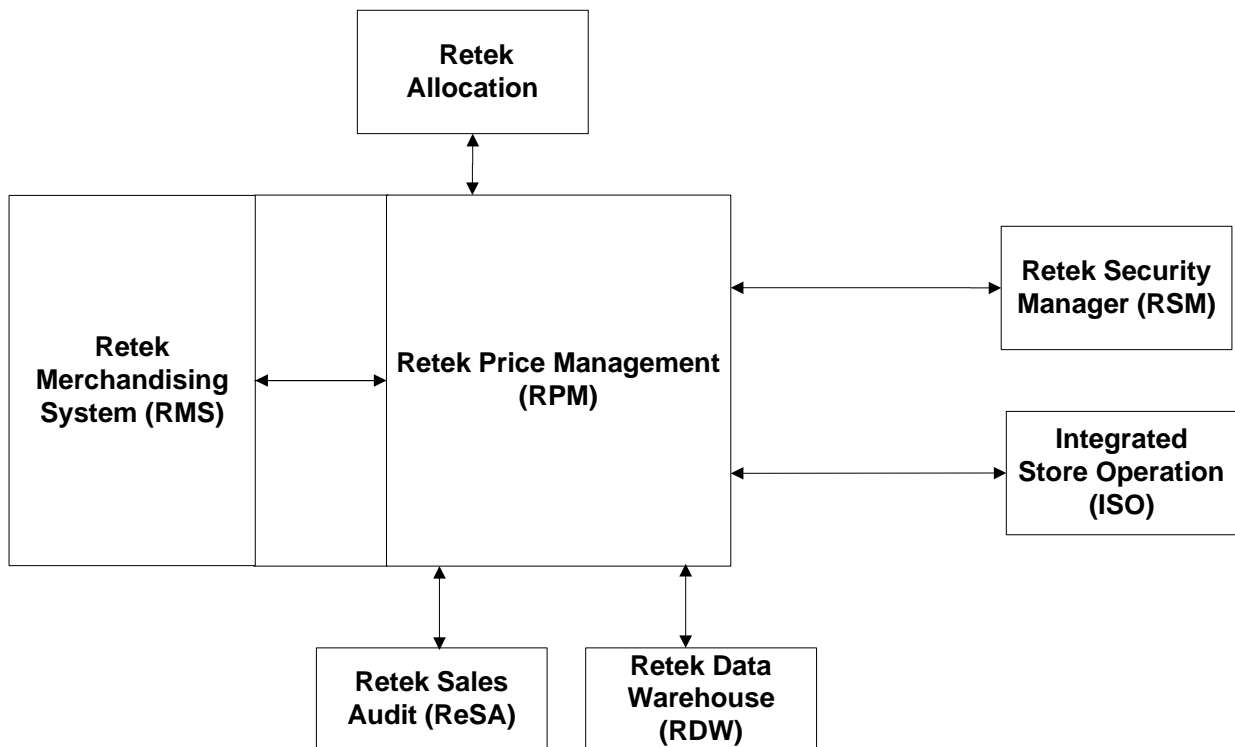
The diagram details the overall direction of the dataflow among the various systems. The accompanying explanations of this diagram are written from a system-to-system perspective, illustrating the movement of item data throughout the RPM-related portion of the enterprise. Note that this discussion focuses on a high-level functional use of data. For a technical description of the dataflow, see “Chapter 5 – Methods of integration”.

### A note about the merchandising system interface

Many tables and functions within RPM are held in common with the Retek 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 Retek product suite is installed) is limited.

### Integration interface dataflow diagram



**RPM-related dataflow across the enterprise**

## Integration interface dataflow description

### From Retek Allocation to RPM

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

### From RPM to Retek Allocation

- Future retail price data  
Retek 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:

- **Store/Warehouse creation**  
This message is used to notify RPM when stores and/or warehouses are added to RMS. RPM needs the new store/warehouse and its associated pricing location in order to assign the new store/warehouse to the zone structure. The message also contains the currency of the new store/warehouse in the event that the pricing location assigned does not share the same currency as the new store/warehouse.
- **Item/Location creation**  
Incoming item/location records create the initial future retail record. RPM also checks to determine whether the new item/location belongs to any approved price change/promotion/clearance, and if so, adds a record to the future retail table for the new item location for each event found.
- **Department creation**  
This message is used to notify RPM when a new department is created in RMS. RPM creates aggregation level information for the new department using predefined system defaults. The user can modify these values via the Maintain Aggregation Levels workflow.

### From RPM to RSM



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

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

### From RSM to RPM

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

### From RPM to ReSA

ReSA needs to receive promotion data from RPM and ReSa retrieves this data via the RETL extract program. Included in this data is promotion detail, promotion events, and promotion headers. For more information regarding this, see ‘RETL program overview for RPM extractions’ in “Chapter 7 – Java and RETL batch processes.”



## From RPM to SIM/ISO and SIM/ISO to RPM

RPM publishes information to Retek Store Inventory Management (SIM), which is an application on the Integrated Store Operations (ISO) platform, 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.
  - ISO 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 ISO of the failure.
  - ISO 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 ISO can apply them to the current regular price or clearance based on the promotion settings. Complex promotions are published with their details, as a specific promotional retail cannot be calculated.
  - SIM creates simple promotions at the item/location level. RPM checks the submitted promotion for conflicts and overlaps. RPM also checks for negative retails to insure that the promo retail is not above the regular retail. Approved promotions are published, and SIM is able to implement the promotion. If RPM validation is unsuccessful, RPM informs SIM of the failure.
  - SIM edits existing simple promotions. RPM validates the edits to ensure they 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 RDW

RDW needs to receive promotion data from RPM and RDW retrieves this data via the RETL extract program. Included in this data is promotion detail, promotion events, and promotion headers. For more information regarding this, see 'RETL program overview for RPM extractions' in "Chapter 7 – Java and RETL batch processes."

## Price management status page

Because RPM is dependent upon a number of servers, and a number of Retek 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 Retek 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?

# Chapter 5 – Methods of integration

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

- Retek Integration Bus (RIB)-based integration
- Retek Service Layer (RSL)-based integration
- Persistence layer integration

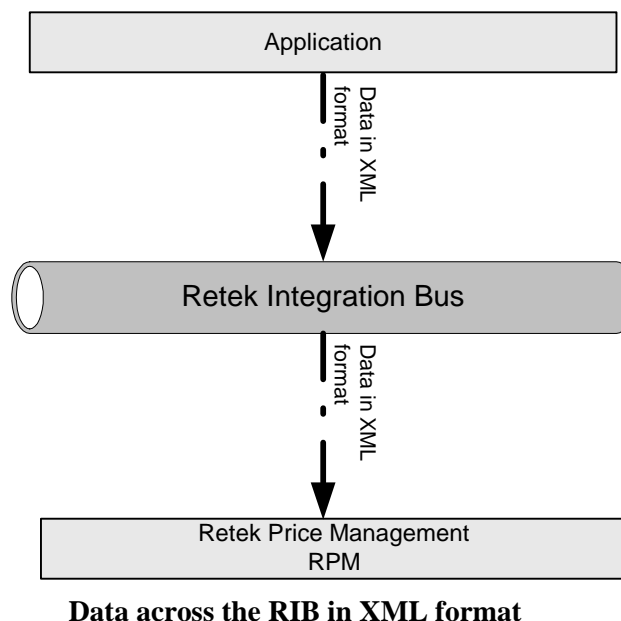
Each section includes information concerning the architecture of the integration method and the data that is being passed back and forth. For additional functional descriptions of the dataflow, see “Chapter 4 – Integration functional dataflows”.

## RPM and the Retek Integration Bus (RIB)

The flow diagrams and explanations in this section provide a brief overview of publication and subscription processing. See the latest Retek Integration documentation for additional information. For information about RIB-related configuration within the RPM application, see the sections ‘rib\_user.properties’ and ‘Disabling RIB publishing in RPM’ in “Chapter 2 – Backend system administration and configuration”.

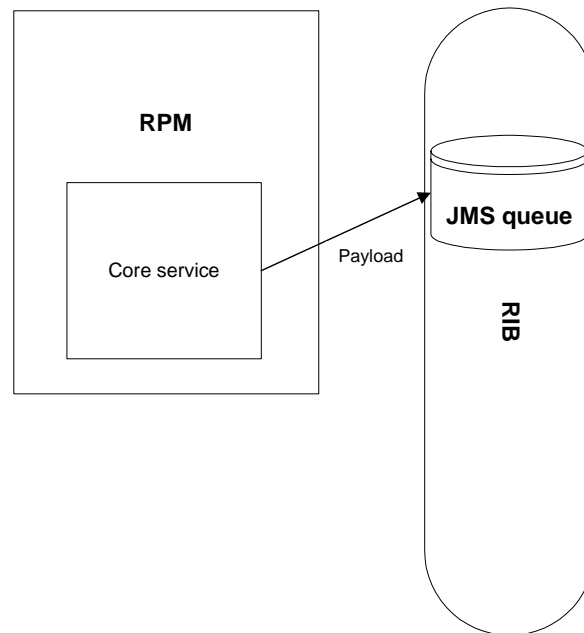
### The XML message format

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



## Message publication processing

As shown by the diagram below, 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.



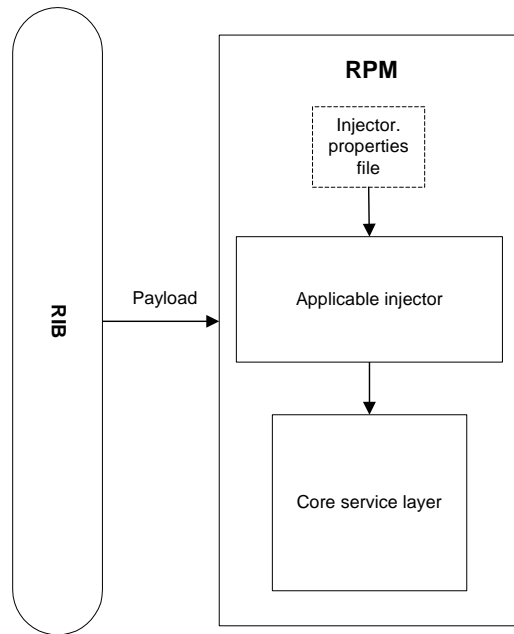
**RPM message publication processing**

## Message subscription processing

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



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

**RPM message subscription processing**

## Publishers mapping table

This table illustrates the relationship among the message family, message type and DTD/payload. For additional information, see the latest Retek Integration documentation.

Family	Type	DTD/Payload
regprcchg	regprcchgcre	RegPrCChgDtl
regprcchg	regprcchgmod	RegPrCChgDtl
regprcchg	regprcchgdel	RegPrCChgDtlRef
clrprcchg	clrprcchgcre	ClrPrCChgDtl
clrprcchg	clrprcchgmod	ClrPrCChgDtl
clrprcchg	clrprcchgdel	ClrPrCChgDtlRef
prmprcchg	prmprcchgcre	PrmPrCChgDtl
prmprcchg	prmprcchgmod	PrmPrCChgDtl
prmprcchg	prmprcchgdel	PrmPrCChgDtlRef

## Subscribers mapping table

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

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

## Functional descriptions of messages

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

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

Functional Area	Subscription/Publication	Integration to products	Description
Item/Location Creation	Subscribe	RMS	This message is used to notify RPM when a new item/location relationship has been created. RPM needs to process this message and add future retail records for the new item/location if any approved price changes/promotions/clearances exist at a parent/zone level that encompasses the new item/location.
Item Modification	Subscribe	RMS	This message is used to notify RPM if there is an Item Reclassification in RMS. RPM needs this information to update the RPM_FUTURE_RETAIL table to have the correct merchandise hierarchy.
Department Creation	Subscribe	RMS	This message is used to notify RPM when a new department is created in RMS. RPM creates and sets default aggregation level data for the new department when the message is processed.
Price Change Creation	Publish	RMS, ISO	This message is used by RPM to communicate the approval of a price change within the application. This message is published at a transaction item/location level.
Price Change Modification	Publish	RMS, ISO	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	Publish	RMS, ISO	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.

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



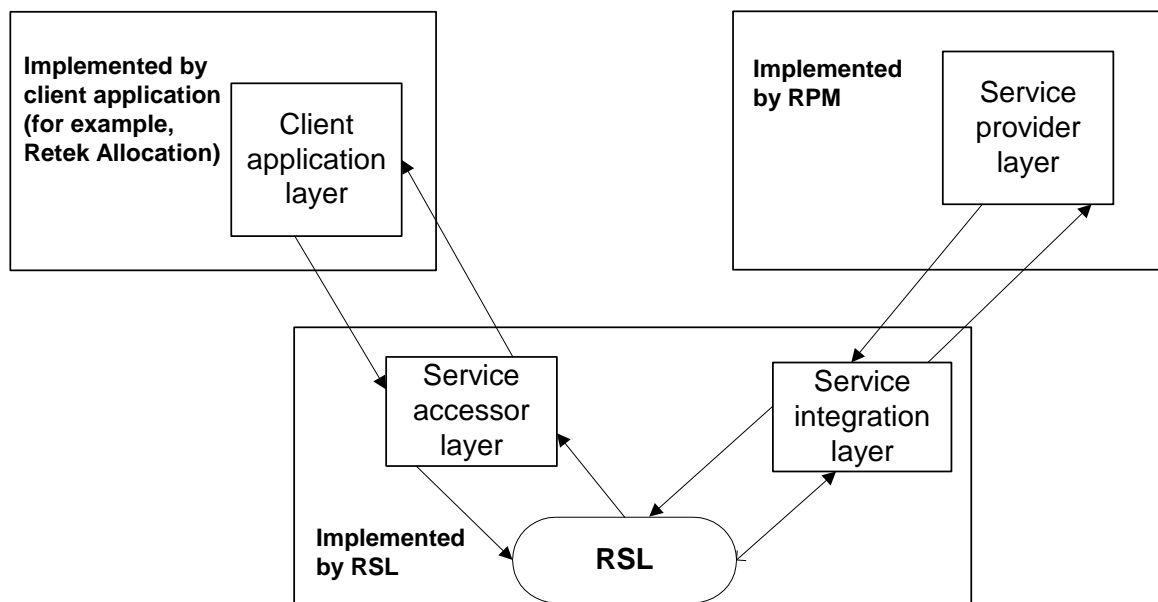
Functional Area	Subscription/Publication	Integration to products	Description
Promotion Deletion	Publish	ISO	This message is used by RPM to communicate the deletion (un-approval included) of an already approved promotion. This message is published at a transaction item/location level, and is used to notify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

## RPM and the Retek Service Layer (RSL)

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

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

For information about RSL-related configuration within the RPM application, see “Chapter 2 – Backend system administration and configuration”.



**Client application and service provider processing through RSL**

## Functional description of the class using RSL

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

Class	Description
PriceInquiryServiceJava.java	This service, provided by RPM, allows an inquiring system to request the effective retail for an item at a specified location on a given date. RPM provides the retail value and indicates whether the value is promotional, clearance or regular.
PriceChangeServiceJava.java	This service, provided by RPM allows a system to request, modify, or delete a price change, clearance, or simple promotion. RPM will reply to this request with a fail/accept status.

## Persistence layer integration

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

- By reading directly from RMS tables.
- By directly calling RMS packages.
- By reading RPM views based on RMS tables.

RMS utilizes RPM packages in order to access processing and information available only in RPM. This type of interaction is only done through package calls.

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

## RMS tables accessed through the persistence layer

RPM uses the tables shown below through the persistence layer:

RMS tables accessed through the persistence layer
AREA
CHAIN
CLASS
CODE_DETAIL
CODE_HEAD
COMP_PRICE_HIST
COMP_STORE
COMPETITOR

<b>RMS tables accessed through the persistence layer</b>
DEAL_COMP_PROM
DEAL_DETAIL
DEAL_HEAD
DEAL_ITEMLOC
DEPS
DIFF_GROUP_DETAIL
DIFF_GROUP_HEAD
DIFF_IDS
DIFF_TYPE
DISTRICT
DIVISION
FUTURE_COST
GROUPS
ITEM_LOC
ITEM_MASTER
ITEM_SEASONS
ITEM_SUPPLIER
LOC_LIST_DETAIL
LOC_LIST_HEAD
PARTNER
PHASE
REGION
SEASONS
SKULIST_DETAIL
SKULIST_HEAD
STORE
SUBCLASS
SUPS
SYSTEM_OPTIONS
UDA
UDA_ITEM_FF
UDA_ITEM_LOV

RMS tables accessed through the persistence layer	
UDA_ITEMDATE	
UDA_VALUES	
UOM_CLASS	
WH	

## RMS packages and methods accessed through RPM's persistence layer

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

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

## RPM views based on RMS tables

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

## RPM packages called by RMS

Packages
MERCH_API_SQL
MERCH_DEALS_API_SQL
MERCH_RETAIL_API_SQL

# Chapter 6 – Functional design

## Overview

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

- Functional assumptions
- Functional overviews
- Concurrency considerations

## Functional assumptions

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

# Functional overviews

## Zone structures

Zone structures in RPM allow 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 item(s) being priced. The following are examples of zone groups:

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

Within zone groups in RPM are groupings of locations (stores and/or warehouses) called 'zones'. The function of these zones is to group locations together in a manner that best facilitates company pricing strategies. 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.

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



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

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



## **Codes**

### **Market basket codes**

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

# Price changes, promotions, clearances, and promotion constraint

## Overview

Pricing events in RPM are broken into three primary categories. Although these pricing activities are unique, they have a common look and feel. The three primary pricing events in RPM are discussed in this section and include:

- Price changes
- Promotions
- Clearances

This section also addresses promotion constraint functionality, where users are warned if they are creating a price change within a set number of days of the start of an approved promotion.

## Price changes

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

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

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

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

## Promotions

Promotions is another pricing event in RPM and while it shares similar characteristics to that of price changes, there are several factors that distinguish promotions from price changes and clearances. The promotion of items is frequently driven by a particular event such as a holiday or the overstock of an item.

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

You also have the option of creating exceptions to promotions created at one of these higher levels (such as, mark all men's turtleneck sweaters down 10%, except mark down the large size 5%). You also have the option to exclude item/locations from a promotion.

Some examples of promotion types in RPM are:

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

## Clearances

Clearances in RPM are 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 item(s) and locations where the clearance is in effect and the discount or set price for the markdown.

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

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



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

## Promotion constraint

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

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

### Pricing strategies

The pricing strategies frontend 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 and competitor strategies
- Clearance
- Competitive
- Margin
- Maintain margin

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

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

### Area differentials and competitor strategies

Area differential pricing allows a buyer to perform the following:

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

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

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

### Layered competitive pricing comparisons in the worksheet

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

The area differential price acts as a guide. The 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

## 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 via the worksheet once a price change proposal has been created.

**Merch Extract calculations**

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

**Market basket margin**

Cost method:

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

Retail method:

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

**Current margin**

Current margin % =

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

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

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

Cost Method

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

Retail Method:

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

**Change by cost change amount**

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

**Examples**

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

### Cost method

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

### Retail method

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

## Price inquiry

Price inquiry 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

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

## Worksheet

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

The worksheet dialog has two main screens: the worksheet status screen and the worksheet detail screen. In the worksheet status screen, a table is displayed with each row of the table representing an individual worksheet. You will be able to access these worksheets and depending on the status of the worksheet and the records within the worksheet, you are able to perform the following:

- Submit
- Approve
- Reject
- Reset
- Delete



**Note:** When an action is taken and applied, that action is only be taken against the detail records within the worksheet that can have that specific action applied. For example, if a detail is in 'approved' status, if the action 'submit' is taken, it will not be applied to the record in 'approved' status.

The Worksheet detail form displays information about the records contained in a given worksheet. Among the information displayed is the proposed price change generated by the merchandise extract program and other information that can assist you in making determinations on 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:

- Identify the work to be processed.
- Extract RMS data into RPM
- Use extracted data to propose retails (based on strategies) and build the worksheets.

The merchandise extract batch program will initially find 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 'Y'es, the merchandise extract should tag those Item/Location records that are pulled into the worksheet with an exception flag if any of the following occur during a review period where exceptions are processed: competitor regular retail price changes, cost changes, and new item/location relationships.

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

Below is a list of reasons why 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 is 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 and RETL batch processes” for additional information about this batch process.

## Calendar

Calendars are set up in RPM for the primary purpose of attaching them to pricing strategies. When 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: 01/01/04

End Date: 01/20/04

Review Period Duration: 3 Days

Days Between Review Periods: 3 Days

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



## **Aggregation level**

Aggregation level functionality is used in RPM to define parameters that vary at the department level. Within this functional area, 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 schedule it to move to a different zone within a zone group on that scheduled date. The process of moving a location contains conflict checking to fix potential pricing problems. 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. Conflict checking involves the following:

- Checking for strategies with the old zone/new zone attached
- Checking for other scheduled location moves for that location
- Checking for promotions at the old zone/new zone that span the location move



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

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

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

## Concurrency considerations

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

### Pessimistic data locking

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

For example, 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 will 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 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 will not be 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 will overwrite data submitted by the first user. With *Last User Wins*, there will be 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 will receive the appropriate business exception message.

Scenario:

Two users have been told to update a pricing strategy. User one enters the strategy and changes the value. User two enters the strategy. Since user one has not saved 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

	Pessimistic Data Locking	Pessimistic Workflow Locking	Last User Wins	Optimistic Data Locking
Clearance Price Changes	√		√	
Price Changes	√		√	
Promotions	√		√	
Future Retail/Conflict Checking	√		√	
Location Moves	√		√	
Worksheet		√		
Aggregation Level			√	
Area Differentials			√	
Calendar			√	
Candidate Rules			√	
Foundation			√	
Initial Price Settings			√	

	Pessimistic Data Locking	Pessimistic Workflow Locking	Last User <i>Wins</i>	Optimistic Data Locking
Pricing Attributes			√	
Pricing Guides			√	
Pricing Strategies			√	
Promotional Funding			√	
Security			√	
System Options			√	
Zone Structure			√	
Link Codes			√	
Merch Extract			√	
Zone Future Retail			√	



# Chapter 7 – Java and RETL batch processes

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

## Java batch processes

This section provides the following:

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

## Java batch process architectural overview

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

Note the following characteristics of RPM's batch processes:

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

## Running a Java-based batch process

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

Each script performs the following internally:

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

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



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

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

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

### Additional notes

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

### Script catalog

Script	Batch program executed
<code>locationMoveBatch.sh</code>	<code>LocationMoveBatch</code>
<code>merchExtractKickOffBatch.sh</code>	<code>MerchExtractKickOffBatch</code>
<code>priceChangeAutoApproveResultsPurgeBatch.sh</code>	<code>PriceChangeAutoApproveResultsPurgeBatch</code>
<code>priceChangePurgeBatch.sh</code>	<code>PriceChangePurgeBatch</code>
<code>priceChangePurgeWorkspaceBatch.sh</code>	<code>PriceChangePurgeWorkspaceBatch</code>
<code>priceEventExecutionBatch.sh</code>	<code>PriceEventExecutionBatch</code>
<code>priceStrategyCalendarBatch.sh</code>	<code>PriceStrategyCalendarBatch</code>
<code>promotionPurgeBatchbatch.sh</code>	<code>PromotionPurgeBatchbatch</code>
<code>purgeExpiredExecutedOrApprovedClearancesBatch.sh</code>	<code>PurgeExpiredExecutedOrApprovedClearancesBatch</code>
<code>purgeLocationMovesBatch.sh</code>	<code>PurgeLocationMovesBatch</code>
<code>purgeUnusedAndAbandonedClearancesBatch.sh</code>	<code>PurgeUnusedAndAbandonedClearancesBatch</code>
<code>worksheetAutoApproveBatch.sh</code>	<code>WorksheetAutoApproveBatch</code>
<code>launchRpmBatch.sh</code>	The retailer does <b>not</b> schedule this script. Other batch programs call this script behind the scenes. Note that this script sets up environment information and takes as a parameter the name of the batch program to run.

### Scheduler and the command line

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

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



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

## Functional descriptions and dependencies

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

Batch processes	Details
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.
PriceChangeAutoApproveResultsPurgeBatch	This batch process deletes old error message from the price change auto approve batch program.
PriceChangePurgeBatch	This batch process deletes past price changes.
PriceChangePurgeWorkspaceBatch	This batch process deletes abandoned price change workspace records.
PriceEventExecutionBatch	This batch process performs the necessary work to start (regular price change, clearance price change, promotions) and end (price change, promotions) pricing events.
PriceStrategyCalendarBatch	This batch process maintains calendars assigned to price strategies.
PromotionPurgeBatch	This batch process deletes old and rejected promotions.
PurgeExpiredExecutedOrApprovedClearancesBatch	This batch process deletes expired clearances in 'Executed' or 'Approved' statuses.
PurgeLocationMovesBatch	This batch process cleans up expired/executed location moves
PurgeUnusedAndAbandonedClearancesBatch	This batch process deletes unused and rejected clearances.
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.

## 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 Retek batch standards. They are executed and terminated in the same manner as other batch processes in the Retek suite of products. The following guidelines describe the return values that RPM’s batch processes utilize:

#### Return values

- **1** – The function completed without error.
- **0** – 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”.

## LocationMoveBatch batch design

### Overview

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

### Usage

The following command runs the LocationMoveBatch job:

```
LocationMoveBatch userid password
```

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

### Detail

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

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

Update FUTURE\_RETAIL table to reflect the location move.

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

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

- RPM\_LOC\_MOVE\_PRC\_CHNG\_EX
- RPM\_LOC\_MOVE\_CLEARANCE\_EX
- RPM\_LOC\_MOVE\_PROMO\_COMP\_DTL\_EX

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

### Assumptions and scheduling notes

LocationMoveBatch must run before the following programs:

- PriceEventExecutionBatch,
- MerchExtractKickOffBatch

### Primary tables involved

- RPM\_FUTURE\_RETAIL

### Threading

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

## MerchExtractKickOffBatch batch design

### Overview

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

### Usage

The following command runs the MerchExtractKickOffBatch job:

```
MerchExtractKickOffBatch userid password
```

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

### Detail

Setup: 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.
- Deal with overlapping strategies. Strategies can be setup at the DEPT, CLASS, or SUBCLASS. MerchExtractKickOffBatch.java ensures that only one strategy affects a given item.
- Start threads based on the values in RPM\_BATCH\_CONTRL for MerchExtractKickOffBatch.java.

Worksheet Creation:

- Call RPM\_EXT\_SQL, a PL/SQL package, to extract RPM information. The package is called at the strategy and RPM\_DEPT\_AGGREGATION. WORKSHEET\_LEVEL. level. It pulls large amounts of data from various RMS tables and populates the RPM\_WORKSHEET\_DATA table.
- Read in the RPM\_WORKSHEET\_DATA records created by RPM\_EXT\_SQL. For each RPM\_WORKSHEET\_DETAIL record perform the following:
  - Use the price strategy to propose a retail value.
  - Apply candidate rules.
  - Apply price guides.

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

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

### **Assumptions and scheduling notes**

The following programs must run before PriceStrategyCalendarBatch:

- PriceStrategyCalendarBatch
- LocationMoveBatch

### **Primary (RPM) tables involved**

- RPM\_WORKSHEET\_STATUS
- RPM\_WORKSHEET\_DATA
- RPM\_STRATEGY
  - RPM\_STRATEGY\_CLEARANCE
  - RPM\_STRATEGY\_CLEARANCE\_MKDN
  - RPM\_STRATEGY\_COMPETITIVE
  - RPM\_STRATEGY\_DETAIL
  - RPM\_STRATEGY\_MARGIN
  - RPM\_STRATEGY\_REF\_COMP
  - RPM\_STRATEGY\_WH
- RPM\_AREA\_DIFF
  - RPM\_AREA\_DIFF\_EXCLUDE
  - RPM\_AREA\_DIFF\_PRIM
  - RPM\_AREA\_DIFF\_WH
- RPM\_CALENDAR
  - RPM\_CALENDAR\_PERIOD

- RPM\_CANDIDATE\_RULE
  - RPM\_CONDITION
  - RPM\_VARIABLE
  - RPM\_VARIABLE\_DEPT\_LINK
- RPM\_PRICE\_GUIDE
  - RPM\_PRICE\_GUIDE\_DEPT
  - RPM\_PRICE\_GUIDE\_INTERVAL

### Threading

MerchExtractKickOffBatch.java is threaded. The RPM\_BATCH\_CONTROL table must include a record for MerchExtractKickOffBatch.java for it to run in threaded mode.

MerchExtractKickOffBatch.java is threaded by strategies and the RPM\_DEPT\_AGGREGATION. WORKSHEET\_LEVEL setting.

### PL/SQL interface point

- Package: RPM\_EXT\_SQL

## PriceChangeAutoApproveResultsPurgeBatch batch design

### Overview

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

### Usage

The following command runs the PriceChangeAutoApproveResultsPurgeBatch job:

```
PriceChangeAutoApproveResultsPurgeBatch userid password
```

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

### Detail

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

### Assumptions and scheduling notes

PriceChangeAutoApproveResultsPurgeBatch can be run ad hoc.

### Primary tables involved

- RPM\_MAINT\_MARGIN\_ERR
- RPM\_MAINT\_MARGIN\_ERR\_DTL

### Threading

This program is not threaded.

# PriceChangePurgeBatch batch design

## Overview

The PriceChangePurgeBatch program deletes past price changes.

## Usage

The following command runs the PriceChangePurgeBatch job:

```
PriceChangePurgeBatch userid password
```

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

## Detail

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

## Assumptions and scheduling notes

PriceChangePurgeBatch can be run ad hoc.

## Primary tables involved

- RPM\_PRICE\_CHANGE

## Threading

This program is not threaded.



## PriceChangePurgeWorkspaceBatch batch design

### Overview

The PriceChangePurgeWorkspaceBatch program deletes abandoned price change workspace records.

### Usage

The following command runs the PriceChangePurgeWorkspaceBatch job:

```
PriceChangePurgeWorkspaceBatch userid password
```

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

### Detail

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

### Assumptions and scheduling notes

PriceChangePurgeWorkspaceBatch can be run ad hoc.

### Primary tables involved

- RPM\_PRICE\_WORKSPACE
- RPM\_PRICE\_WORKSPACE\_DETAIL

### Threading

This program is not threaded.

# PriceEventExecutionBatch batch design

## Overview

The price event execution batch process (PriceEventExecutionBatch.java) performs the necessary work to start (regular price change, clearance price change, promotions) and end (price change, promotions) pricing events.

## Usage

The following command runs the job:

```
PriceEventExecutionBatch userid password
```

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

## Detail

The batch program processes regular price changes, clearance price changes, and promotions events that are scheduled for the run date.

- Promotions:
  - Promotions that are scheduled to start are activated.
  - Promotions that are scheduled to end are completed.
- Clearances:
  - Clearance markdowns that are scheduled to take place are executed.
  - Clearances that are scheduled to be completed (reset) are completed.
- Regular price changes:
  - Regular price changes that are scheduled to take place are executed.

## Assumptions and scheduling notes

This batch process must run before the following programs:

- Storeadd (RMS)
- MerchExtractKickOffBatch

The following programs must run before this batch process:

- Salstage (RMS)
- LocationMoveBatch

## Primary tables involved

- RPM\_PRICE\_CHANGE
- RPM\_CLEARANCE
- RPM\_PROMO\_COMP\_DETAIL

### **RMS interface point**

The price event execution batch interfaces with the RMS price change subscription package RMSSUB\_PRICE\_CHANGE. All price change, clearance, and promotion prices will be passed along to this RMS package at the item location level and will be applied in RMS.

### **Threading**

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

## PriceStrategyCalendarBatch batch design

### Overview

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

### Usage

The following command runs the PriceStrategyCalendarBatch job:

```
PriceStrategyCalendarBatch userid password
```

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

### Detail

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

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

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

### Assumptions and scheduling notes

PriceStrategyCalendarBatch must run before the following programs:

- PriceEventExecutionBatch,
- MerchExtractKickOffBatch

### Primary tables involved

- RPM\_STRATEGY
- RPM\_CALENDAR
- RPM\_CALENDAR\_PERIOD

### Threading

This program is not threaded.

## PromotionPurgeBatchbatch design

### Overview

The PromotionPurgeBatch program deletes old and rejected promotions.

### Usage

The following command runs the PromotionPurgeBatch job:

```
PromotionPurgeBatch userid password
```

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

### Detail

PromotionPurgeBatch deletes promotions based on two system options.

RPM\_SYSTEM\_OPTIONS.PROMOTION\_HIST\_MONTHS and

RPM\_SYSTEM\_OPTIONS.REJECT\_HOLD\_DAYS\_PROMO.

RPM\_SYSTEM\_OPTIONS.PROMOTION\_HIST\_MONTHS controls how long non-rejected promotions are held. RPM\_SYSTEM\_OPTIONS.REJECT\_HOLD\_DAYS\_PROMO controls how long rejected promotions are held.

### Assumptions and scheduling notes

PromotionPurgeBatch can be run ad hoc.

### Primary tables involved

- RPM\_PROMO\_EVENT
- RPM\_PROMO
- RPM\_PROMO\_DEAL\_LINK
- RPM\_PROMO\_COMP
- RPM\_PROMO\_COMP\_THRESH\_LINK
- RPM\_PROMO\_DEAL\_COMP\_LINK
- RPM\_PROMO\_COMP\_DETAIL
- RPM\_PROMO\_COMP\_SIMPLE
- RPM\_PROMO\_COMP\_THRESHOLD
- RPM\_PROMO\_COMP\_BUY\_GET
- RPM\_PROMO\_BUY\_GET\_ZONE
- RPM\_PROMO\_BUY\_ITEM
- RPM\_PROMO\_GET\_ITEM
- RPM\_PENDING\_DEAL
- RPM\_PENDING\_DEAL\_DETAIL

### Threading

This program is threaded.

## PurgeExpiredExecutedOrApprovedClearancesBatch batch design

### Overview

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

### Usage

The following command runs the PurgeExpiredExecutedOrApprovedClearancesBatch job:

```
PurgeExpiredExecutedOrApprovedClearancesBatch userid password
```

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

### Detail

The PurgeExpiredExecutedOrApprovedClearancesBatch deletes clearances that meet the following criteria:

- Clearance effective date is older than the CLEARANCE\_HIST\_MONTHS system option.
- Clearance is on a valid future retail timeline (RPM\_FUTURE\_RETAIL).
- Clearance is in an Approved or Executed status.

### Assumptions and scheduling notes

PurgeExpiredExecutedOrApprovedClearancesBatch can be run ad hoc.

### Primary tables involved

- RPM\_CLEARANCE

### Threading

This program is not threaded.

## PurgeLocationMovesBatch batch design

### Overview

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

### Usage

The following command runs the PurgeLocationMovesBatch job:

```
PurgeLocationMovesBatch userid password
```

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

### Detail

The PurgeLocationMovesBatch program deletes location moves based on their effective date. Location moves are purged regardless whether or not they have been executed. Location moves are purged when their effective date is RPM\_SYSTEM\_OPTIONS.LOCATION\_MOVE\_PURGE\_DAYS days in the past.

### Assumptions and scheduling notes

PurgeLocationMovesBatch can be run ad hoc.

### Primary tables involved

- RPM\_LOCATION\_MOVE
- RPM\_LOC\_MOVE\_PROMO\_ERROR
- RPM\_LOC\_MOVE\_PRC\_STRT\_ERR
- RPM\_LOC\_MOVE\_PRC\_CHNG\_EX
- RPM\_LOC\_MOVE\_CLEARANCE\_EX
- RPM\_LOC\_MOVE\_PROMO\_COMP\_DTL\_EX

### Threading

PurgeLocationMovesBatch.java is not threaded.

# PurgeUnusedAndAbandonedClearancesBatch batch design

## Overview

The PurgeUnusedAndAbandonedClearancesBatch program deletes unused and rejected clearances.

## Usage

The following command runs the PurgeUnusedAndAbandonedClearancesBatch job:

```
PurgeUnusedAndAbandonedClearancesBatch userid password
```

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

## Detail

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

- Clearance effective date is older than the CLEARANCE\_HIST\_MONTHS system option.
- Clearance is in 'Worksheet' or 'Submitted' status.

Or

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

Or

- Clearance effective date is older than the REJECT\_HOLD\_DAYS\_PC\_CLEAR system option.
- Clearance is in 'Rejected' status.

## Assumptions and scheduling notes

PurgeUnusedAndAbandonedClearancesBatch can be run ad hoc.

## Primary tables involved

- RPM\_CLEARANCE

## Threading

This program is not threaded.



## WorksheetAutoApproveBatch batch design

### Overview

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

### Usage

The following command runs the WorksheetAutoApproveBatch job:

```
WorksheetAutoApproveBatch userid password
```

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

### Detail

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

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

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

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

Each worksheet detail that meets the above criteria will have run through the approval logic. The approval logic will attempt 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 will also be processed. If the secondary area is marked as auto approve, the secondary worksheet detail record will go 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 have a retail proposed for it and will move into new status and become available for review by online users.

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

### Assumptions and scheduling notes

WorksheetAutoApproveBatch can be run ad hoc.

### Primary tables involved

- RPM\_STRATEGY\_MAINT\_MARGIN
- RPM\_WORKSHEET\_STATUS
- RPM\_WORKSHEET\_DATA
- RPM\_PRICE\_CHANGE
- RPM\_CLEARANCE
- RPM\_FUTURE\_RETAIL
- RPM\_AREA\_DIFF\_PRIM
- RPM\_AREA\_DIFF
- RPM\_MAINT\_MARGIN\_ERR
- RPM\_MAINT\_MARGIN\_ERR\_DTL

### Threading

This program is threaded by strategy. Each strategy that is found is processed in its own thread (when threading is set to run on RPM\_BATCH\_CONTROL).

## RETL program overview for RPM extractions

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

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

These extractions were initially designed for Retek Data Warehouse (RDW) but can be used for some other application in the retailer's enterprise.

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

### Architectural design

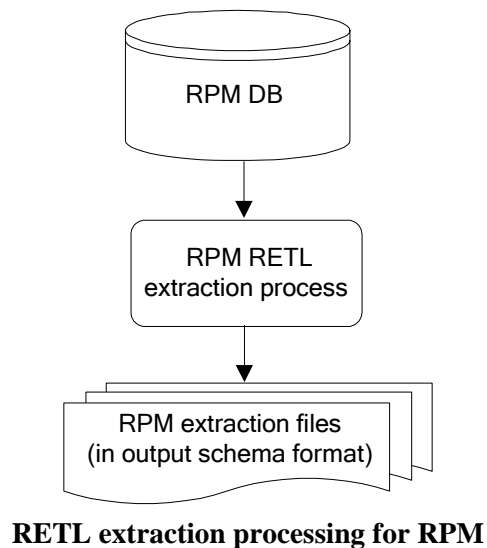
The diagram below illustrates the extraction processing architecture for RPM. Instead of managing the change captures as they occur in the source system during the day, the process involves extracting the current data from the source system. The extracted data is output to flat files. These flat files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

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

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

## RPM extraction architecture

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



## Configuration

### RETL

Before trying to configure and run RPM RETL, install RETL version 11.2 or later, which is required to run RPM RETL. Run the 'verify\_retl' script (included as part of the RETL installation) to ensure that RETL is working properly before proceeding.

### RETL user and permissions

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

### Environment variables

See the RETL Programmer's Guide for RETL environment variables that must be set up for your version of RETL. You will need to set MMHOME to your base directory for RPM RETL. This is the top level directory that you selected during the installation process. In your .kshrc, you should add a line such as the following:

```
export MMHOME=<base directory for RMS ETL>\dwi11.0\dev
```



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

## dwi\_config.env settings

Make sure to review the environmental parameters in the `dwi_config.env` file before executing batch modules. There are several variables you must change depending upon your local settings:

For example:

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

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

```
export PASSWORD=mypasswd
```

### Steps to configure RETL

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



**Note:** All RPM tables have to be under the RMS database. It has the same `BA_OWNER` as RMS. Thus, the only piece that RPM must modify in the `dwi_config.env` file is to assign a value to `RPM_OWNER`. The configuration file, `dwi_config.env`, as well as all other configure files are packed with RMS.

### Program features

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

### Program status control files

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

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

### File naming conventions

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

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

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

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

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

### Restart and recovery

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

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

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

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

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

## Message logging

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

### Daily log file

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

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

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

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

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

### Format

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

```
prmevtex 15:47:14: Program started...
```

```
prmevtex 15:47:18: Program completed successfully
```

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

### Program error file

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

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

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

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

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

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

### Schema files

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

### Resource files

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

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

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



**Note:** Resource files are only packed with RMS.

### Typical run and debugging situations

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

For example:

To run prmevtex.ksh:

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

```
%prmevtex.ksh
```

If the module runs successfully, the following results:

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



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

1. **Log file:** Today's log file, 20040309.log, does not contain the "Program completed successfully" message for prmevtex.ksh.
2. **Data:** The prmevtdm.txt file may exist in the data directory but may not contain all the extracted records.
3. **Error file:** The program's error file, prmevtex.txt.20040309, may contain an error message.
4. **Program status control:** The program status control file, prmevtex.status.20040309, exists.

To re-run a module from the beginning, perform the following actions:

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

```
%prmevtex.ksh
```



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

## RETL extractions program list

This section serves as a reference to the RETL extraction RPM programs.

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

## RETL extract program flow diagrams

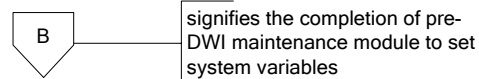
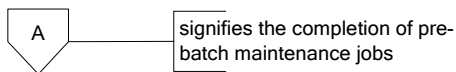
This section presents flow diagrams for data processing. The source system's program or output file is illustrated along with the program or process that interfaces with the source.

Before setting up a program schedule, familiarize yourself with the functional and technical constraints associated with each program.

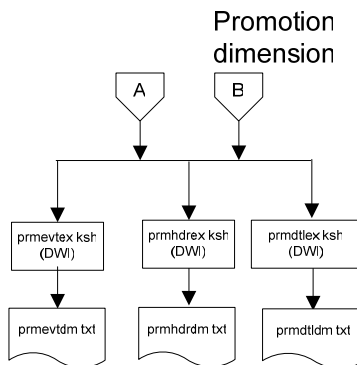
### Legend



**Note:** See the RMS Operations Guide for more information regarding the modules shown in the legend below.



### Program flow diagram



## Application programming interface (API) flat file specifications

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

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

### API format

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

## File layout

- **Field Name:** Provides the name of the field in the text file.
- **Description:** Provides a brief explanation of the information held in the field.
- **Data Type/Bytes:** Includes both data type and maximum column length. Data type identifies one of three valid data types: character, number, or date. Bytes identifies the maximum bytes available for a field. A field may not exceed the maximum number of bytes (note that ASCII characters usually have a ratio of 1 byte = 1 character)
  - **Character:** Can hold letters (a,b,c...), numbers (1,2,3...), and special characters (\$,#,&...)
  - **Numbers:** Can hold only numbers (1,2,3...)
  - **Date:** Holds a specific year, month, day combination. The format is “YYYYMMDD”, unless otherwise specified.
- **Any required formatting for a field is conveyed in the Bytes section.** For example, Number(18,4) refers to number precision and scale. The first value is the precision and always matches the maximum number of digits for that field; the second value is the scale and specifies, of the total digits in the field, how many digits exist to the right of the decimal point. For example, the number –12345678901234.1234 would take up twenty ASCII characters in the flat file; however, the overall precision of the number is still (18,4).
- **Field Order:** Identifies the order of the field in the schema file.
- **Required Field:** Identifies whether the field can hold a null value. This section holds either a ‘yes’ or a ‘no’. A ‘yes’ signifies the field may not hold a null value. A ‘no’ signifies that the field may, but is not required, to hold a null value.

## General business rules and standards common to all APIs

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

If there are no records for the day, an empty flat file must still be provided.

- **Updated and new records of (what RDW refers to as fact data):**  
Facts being loaded to RDW can either be new or updated facts. Unlike dimension snapshots, fact flat files will only contain new/updated facts exported from the source system once per day (or week, in some cases). Refer to each API’s specific business rules section for more details.

If there are no new or changed records for the day, an empty flat file must still be provided.

- **Primary and local currency amount fields**  
Amounts will be stored in both primary and local currencies for most fact tables. If the source system uses multi-currency, then the primary currency column holds the primary currency amount, and the local currency column holds the local currency amount. If the location happens to use the primary currency, then both primary and local amounts hold the primary currency amount. If the source system does not use multi-currency, then only the primary currency fields are populated and the local fields hold NULL values.
- **Leading/trailing values:**  
Values entered into the text files are the exact values processed and loaded into the datamart tables. Therefore, the values with leading and/or trailing zeros, characters, or nulls are processed as such. RDW does not strip any of these leading or trailing values, unless otherwise noted in the individual API's business rules section.
- **Indicator columns:**  
Indicator columns are assumed to hold one of two values, either "Y" for yes or "N" for no.
- **Delimiters:**



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

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

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

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

```
WIS;20010311;1.73527820592648544918
```

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

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

```
1|2|3|4
```

```
5|6|7|8
```

```
9|10|11|12
```

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

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

**prmdtldm.txt**

Business rules:

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

Name	Description	Data Type/Bytes	Field order	Required field
PRMTN_DTL_IDNT	The unique identifier of a promotion detail.	VARCHAR2(10)	1	Yes
HEAD_IDNT	The unique identifier of a promotion head.	VARCHAR2(10)	2	Yes
EVENT_IDNT	The unique identifier of a promotion event.	VARCHAR2(10)	3	Yes
PRMTN_TRIG_TYPE_IDNT	The unique identifier of the promotion trigger type. Valid values can be 'offer code', 'media code', and so on.	NUMBER(10)	4	Yes
PRMTN_SRC_CDE	The unique identifier of a promotion source. Valid values can be 'DTC', 'RPM' or any other promotion source chosen by client.	VARCHAR2(6)	5	Yes
PRMTN_SVC_TYPE_IDNT	The unique identifier of a promotion service type.	VARCHAR2(10)	6	Yes
PRMTN_FMT_IDNT	The unique identifier of a promotion format.	VARCHAR2(10)	7	Yes
BEG_DT	The date the promotion begins.	DATE	8	Yes
PRMTN_DTL_DESC	Description for the promotion detail identifier.	VARCHAR2(160)	9	No

Name	Description	Data Type/Bytes	Field order	Required field
PRMTN_SVC_TYPE_DESC	Description for the promotion service type.	VARCHAR2(120)	10	No
PRMTN_FMT_DESC	Description for the promotion format.	VARCHAR2(120)	11	No
END_DT	The date the promotion ends.	DATE	12	No

### prmevtdm.txt

Business rules:

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

Name	Description	Data Type/Bytes	Field order	Required field
EVENT_IDNT	The unique identifier of a promotion event.	VARCHAR2(10)	1	Yes
EVENT_DESC	Description for the promotion event.	VARCHAR2(255)	2	No
THEME_DESC	Description for the promotion theme.	VARCHAR2(120)	3	No

**prmhdrdm.txt**

Business rules:

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

Name	Description	Data Type/Bytes	Field order	Required field
HEAD_IDNT	The unique identifier of a promotion head.	VARCHAR2(10)	1	Yes
EVENT_IDNT	The unique identifier of a promotion event.	VARCHAR2(10)	2	Yes
HEAD_NAME	Name for the promotion head.	VARCHAR2(120)	3	No
HEAD_DESC	Description for the promotion head.	VARCHAR2(160)	4	No
BEG_DT	The date the promotion begins.	DATE	5	Yes
END_DT	The date the promotion ends.	DATE	6	No