

# Retek<sup>®</sup> Price Management<sup>™</sup> 11.0

## 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.

Retek<sup>®</sup> Price Management<sup>™</sup> 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This unpublished work is protected by confidentiality agreement, and by trade secret, copyright, and other laws.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:

©2004 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@retex.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
A note about the online help .....	3
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 .....	5
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 file .....	9
Hibernate logging .....	9
Internationalization and localization .....	10
Translation .....	10

<b>Chapter 3 – Technical architecture .....</b>	<b>11</b>
Overview .....	11
The layered model .....	12
Client .....	13
Application services layer (stateless session beans) .....	13
Core services layer .....	14
Persistence layer .....	14
Database layer .....	14
Security .....	14
RPM-related Java terms and standards .....	15
<b>Chapter 4 – Integration functional dataflows .....</b>	<b>17</b>
A note about the merchandising system interface .....	17
Integration interface dataflow diagram .....	17
Integration interface dataflow description .....	18
From Retek Allocation to RPM .....	18
From RPM to Retek Allocation .....	18
From RPM to RMS .....	18
From RMS to RPM .....	19
From RPM to RSM .....	19
From RSM to RPM .....	19
<b>Chapter 5 – Methods of integration .....</b>	<b>21</b>
RPM and the Retek Integration Bus (RIB) .....	21
The XML message format .....	21
Message publication processing .....	22
Message subscription processing .....	22
Publishers mapping table .....	23
Subscribers mapping table .....	23
Functional descriptions of messages .....	24
Persistence layer integration .....	25
RMS tables accessed through the persistence layer .....	26
Packages and methods accessed through the persistence layer .....	27
RPM and the Retek Service Layer (RSL) .....	28
Functional description of the class using RSL .....	28

<b>Chapter 6 – Functional design .....</b>	<b>29</b>
Overview .....	29
Functional assumptions .....	29
Functional overviews .....	30
Zone structures .....	30
Price changes, promotions, clearances .....	31
<b>Chapter 7 – Java batch processes.....</b>	<b>33</b>
Java batch process architectural overview .....	33
Running a Java-based batch process .....	33
Scheduler and the command line .....	33
Java packages and their main class .....	34
Functional descriptions and dependencies .....	34
Batch process scheduling .....	34
Return value batch standards .....	34
Return values .....	34
Batch logging .....	35
PriceStrategyCalendarBatch batch design .....	35
Overview .....	35
Assumptions and scheduling notes .....	35
Primary tables involved .....	35
PriceEventExecutionBatch batch design .....	36
Overview .....	36
Assumptions and scheduling notes .....	36
Primary tables involved .....	36
RMS interface point .....	37
LocationMoveBatch batch design .....	37
Overview .....	37
Assumptions and scheduling notes .....	37
Primary tables involved .....	37





# 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 dialogues for creating and maintaining regular price changes, clearances, and promotions. Although each of the three pricing activities is unique, the system displays these dialogues using a common look and feel. Each of these dialogues 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

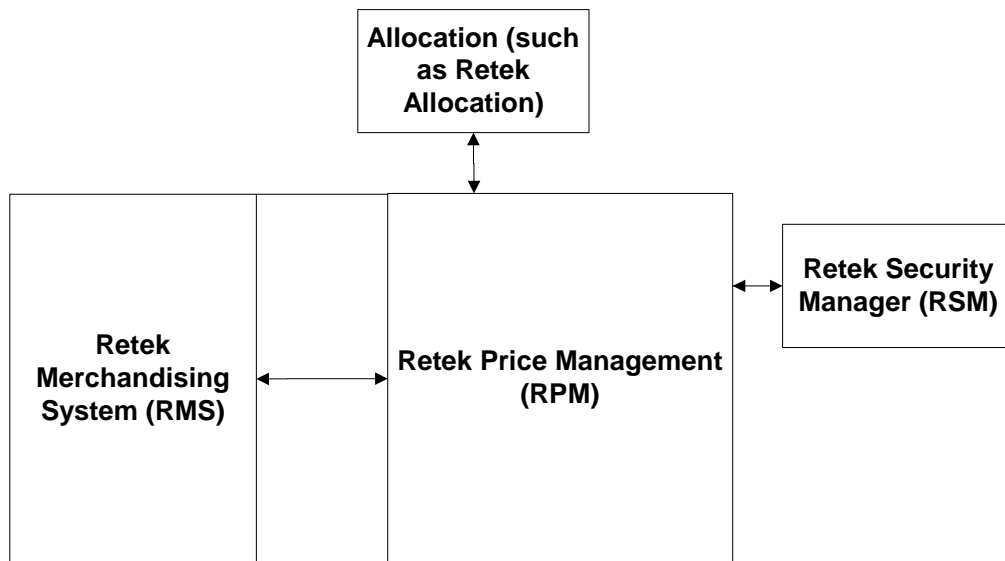
RPM's architecture is built upon a layered model. That is, layers of the application communicate with one another through an established hierarchy and are only able to communicate with neighboring layers. Any given layer need not be concerned with the internal functional tasks of any other layer.

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**

## A note about the online help

The online help for this release of RPM is being delivered as a .CHM file. The .CHM file is a standalone Help, which requires Internet Explorer to run. The .CHM file contains all of the application's HTML help files in a compressed and compiled format.

The RPM retailer receives a deliverable called `Retek_Price_Management.chm` in the same folder as other documentation deliverables (along with the User Guide, Installation Guide, and so on).

## 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 .chm 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.x
- Retek Allocation 11.x
- Retek Security Manager (RSM) 11.x
- RIB 11.x

## 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.

### **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/rettek`. 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”.

```
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.rs1.rpm">
      <impl package="com.retek.rs1.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".

# 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 file

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>
```

# Internationalization and localization

The technical infrastructure of RPM supports languages other than English. RPM has been subject to the modifications associated with ‘internationalization’, also known as I18N. (The I18N name stems from the fact that eighteen letters exist between the first ‘i’ and the last ‘n’ in the word ‘internationalization.’) Internationalization is the process of preparing software in order to ensure that it can efficiently handle multiple languages. In other words, the software is created so that it can be released into international markets.

Localization, also known as L10N, is the process of adapting software that has been internationalized so that it can be released into a local market with its own language. (The L10N name stems from the fact that ten letters separate the letter ‘l’ from the letter ‘n’ in the word ‘localization’.) Software is only internationalized once. However, software must undergo the localization process for every new language or location into which it is released.

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

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

## 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.

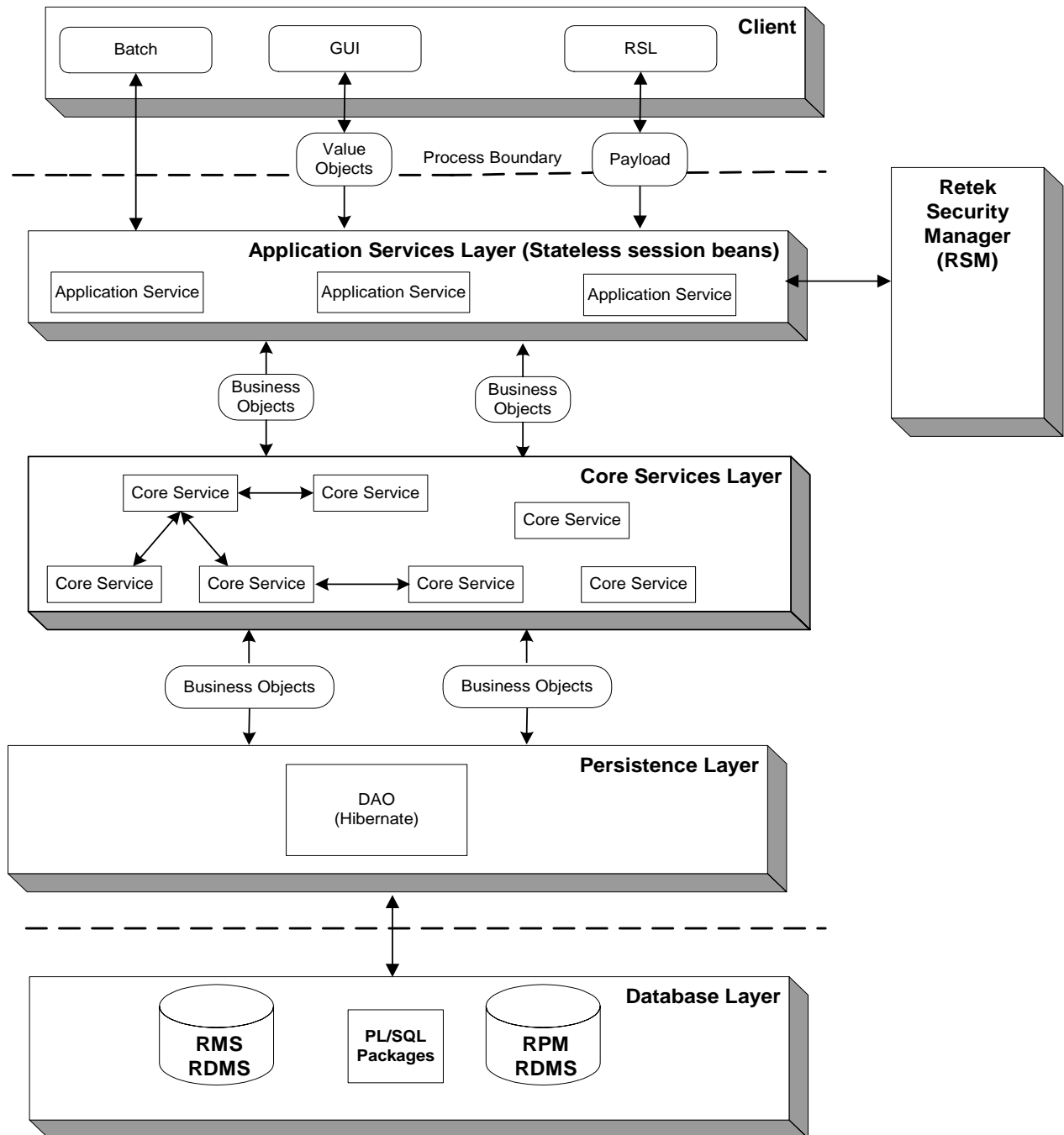
Conceptually, RPM's J2EE architecture is built upon 4-layers and implements what is defined as a service-oriented architecture. Such an architecture is essentially a collection of services that pass data, perform business processing, coordinate system activities, and render data into abstract objects.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

## 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).

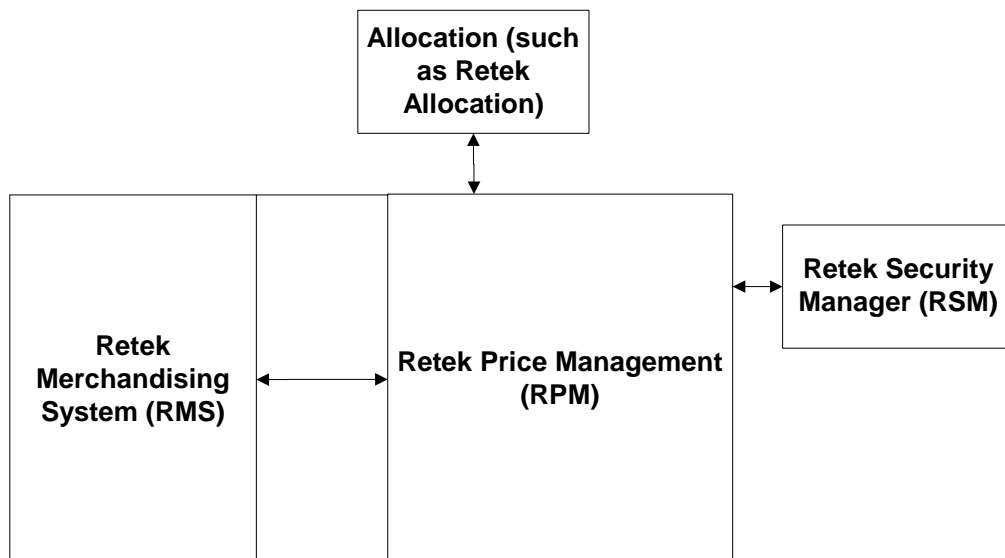
The diagram details the overall direction of the dataflow among the various systems. The accompanying explanations of this diagram are written from a system-to-system perspective, illustrating the movement of item data throughout the RPM-related portion of the enterprise. Note that this discussion focuses 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 allocation (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.
- Promotion execution  
RPM updates promotional retail information in RMS when the promotion is executed in RPM (for example, 25% of the retail price of an item). 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 updates the retail information stored in RMS to reflect the new clearance retail prices determined by RPM.



**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 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/warehouse in the event that the pricing location assigned does not share the same currency as the new store/warehouse.
- Item/Location creation  
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 that now contain this new item/location (for example, price changes set up at the zone level).
- 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).



# 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

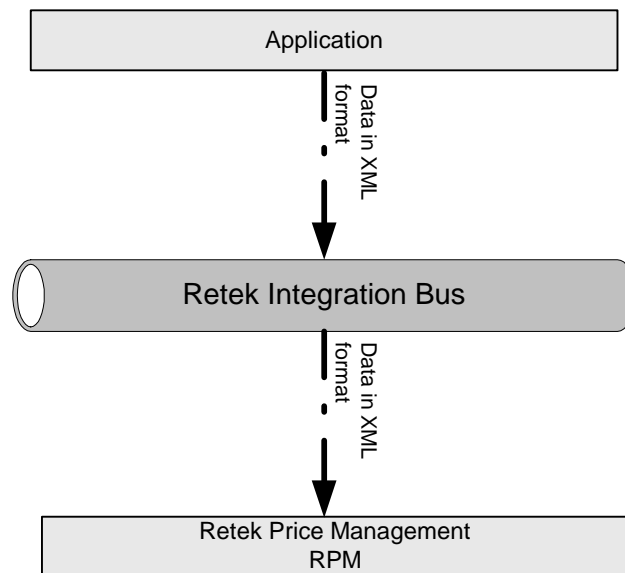
Each section includes information concerning the architecture of the integration method and the data that is being passed back and forth. For additional functional descriptions of the dataflow, see “Chapter 4 – Integration functional dataflows”.

## 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 “Chapter 2 – Backend system administration and configuration”.

### The XML message format

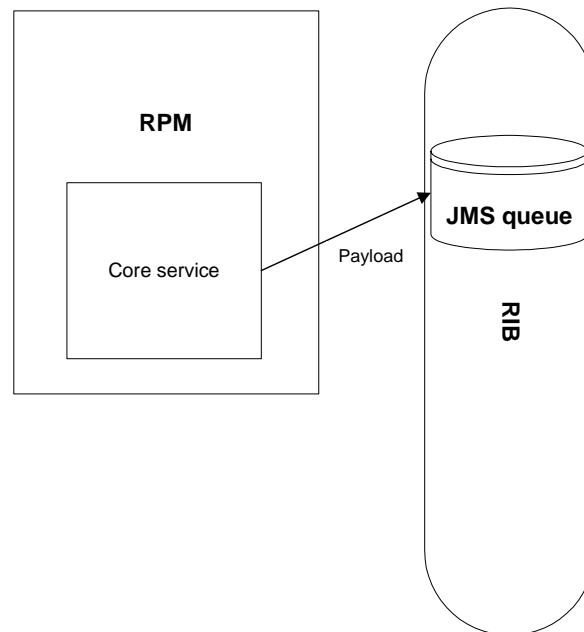
As shown by the diagram below, the messages to which RPM subscribes and which RPM publishes are in an XML format and have their data structure defined by document type definitions (DTDs) or XML schema documents.



**Data across the RIB in XML format**

### Message publication processing

As shown by the diagram below, an event within 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.



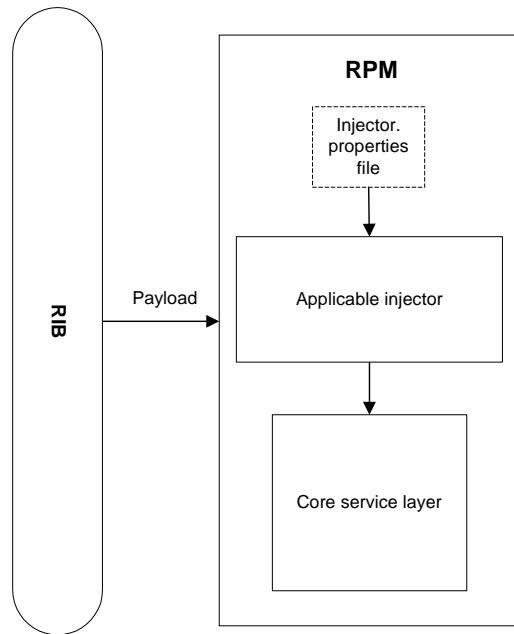
**RPM message publication processing**

### 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

**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.
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.
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.



Functional Area	Subscription/Publication	Integration to products	Description
Price Change Creation	Publish	RMS	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	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	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.

## 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 through the persistence with RMS in two ways:

- By reading directly from RMS tables.
- By directly calling RMS packages.

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:

<b>RMS tables accessed through the persistence layer</b>
SYSTEM_OPTIONS
AREA
CHAIN
CLASS
COMP_STORE
COMPETITOR
DEPS
DIFF_IDS
DIFF_TYPE
DISTRICT
ITEM_MASTER
SKULIST_HEAD
SKULIST_DETAIL
ITEM_LOC
LOC_LIST_HEAD
LOC_LIST_DETAIL
PARTNER
PHASE
REGION
CODE_HEAD
CODE_DETAIL
ITEM_SEASONS
STORE
SUBCLASS
SUPS
UOM_CLASS
ITEM_SUPPLIER
UDA

<b>RMS tables accessed through the persistence layer</b>
UDA_VALUES
WH
DIFF_GROUP_DETAIL
DIFF_GROUP_HEAD
DEAL_HEAD
DEAL_DETAIL
DIVISION
GROUP
DEAL_ITEMLOC
FUTURE_COST
COMP_PRICE_HIST
DEAL_COMP_PROM
ITEM_ZONE_PRICE
UDA_ITEMDATE
UDA_ITEM_FF
UDA_ITEM_LOV

### **Packages and methods accessed through the persistence layer**

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

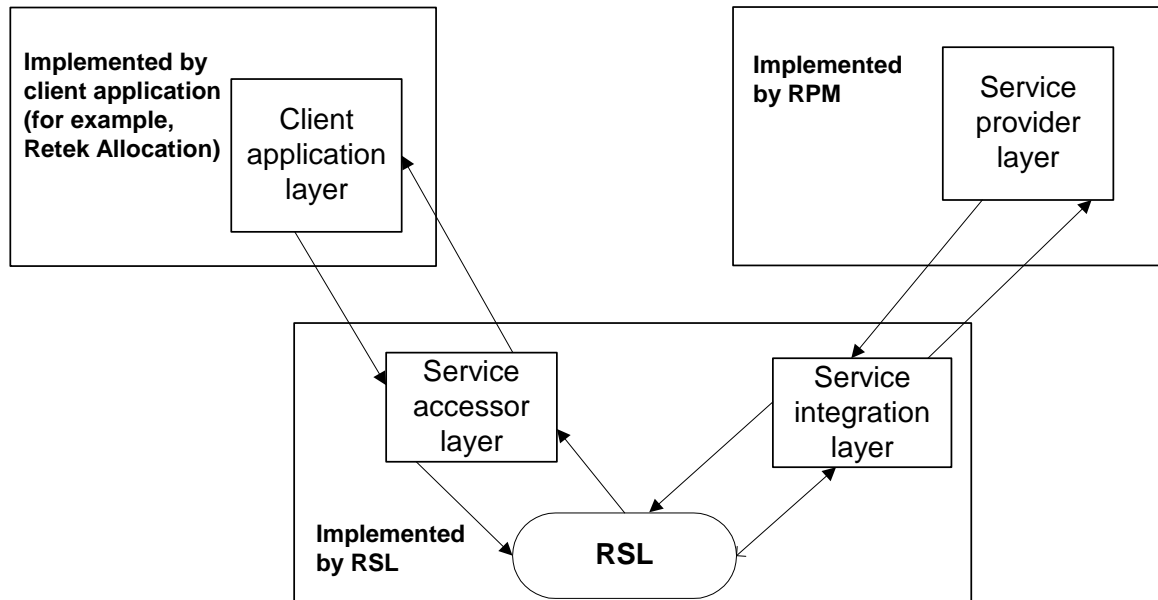
<b>RMS Packages</b>	<b>RMS Methods</b>
RPM_WRAPPER	uom_convert_value
	valid_uom_for_items
	get_vat_rate_include_ind
	currency_convert_value
PM_DEALS_API_SQL	create_deal
	new_deal_comp
RMSSUB_PRICECHANGE	get_price_change

## RPM 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
PriceInquiryJava.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.

# Chapter 6 – Functional design

## Overview

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

- Functional assumptions
- Functional overviews

## Functional assumptions

- Initial price setting does not respect link codes.
- Area differentials are not triggered by new item/location relationships (that is, if the user creates the secondary area item/location relationship and then creates the primary area item/location relationship, the area differential price changes is not created to respect the area differential requested on the strategy).
- Initial price setting does not respect area differentials.
- 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.
- RMS continues to contain one pricing group. This pricing group is created (given a hard coded ID and description) during the installation of RMS. The zone group is at the store level. For RMS and RPM to be integrated successfully, this pricing group data must exist in RMS.
- 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).

## Functional overviews

### Zone structures

Zone structures in RPM allows 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 may also be flexibly defined. For example, you may choose to create zones based on geographic regions such as the following:

- US East region
- US West region
- Mexico stores

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

- US urban stores
- US rural stores

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

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

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.

## **Price changes, promotions, clearances**

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:

- Price changes
- Promotions
- Clearance

### **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 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, RPM updates the RMS pricing information 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 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 a markdown or a series of markdowns designed to increase demand and therefore move inventory out of a store. Subsequent clearances will 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 will be 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. They can be applied at the location or price zone level.

When creating clearances in RPM, the discount used to derive the clearance price is not retained after the price goes into effect. The clearance price is sent to the stores but not the discount used to derive it. In addition to this, when a clearance price is created, you can specify a reset date in which the clearance price will revert back to regular retail price. Reset dates are optional.



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



# Chapter 7 – Java batch processes

This chapter 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 java programs (.java files). Retek provides each of these files.

For example:

```
LocationMoveBatch.java
```

## 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.

## Java packages and their main class

The following table describes the executable shell scripts and Java packages along with the main class within them that defines the (batch) Java class that runs.

Java package	Main class
com.retek.rpm.batch.LocationMove	LocationMoveBatch
com.retek.rpm.batch	PriceEventExecutionBatch
com.retek.rpm.batch	PriceStrategyCalendarBatch

## 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.
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.

## Batch process scheduling

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

## 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

- **Success** - The function completed without error.
- **Failure** - 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”.

## 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

## 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**

Package: RMSSUB\_PRICECHANGE

**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.

**Assumptions and scheduling notes**

LocationMoveBatch must run before the following programs:

- PriceEventExecutionBatch,
- MerchExtractKickOffBatch

**Primary tables involved**

RPM\_ZONE\_LOCATION