

Retek® Price Management 10.1



DAL Porting Guide



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.

Corporate Headquarters:

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

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

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This unpublished work is protected by confidentiality agreement, and by trade secret, copyright, and other laws.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:

©2002 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 e-mail, 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.

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

Internet (ROCS)	www.retek.com/support Retek's secure client Web site to update and view issues
------------------------	------------------------------------------------------------------------------------------------------------------------------

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

Phone	US & Canada: 1-800-61-RETEK (1-800-617-3835) World: +1 612-587-5800 EMEA: 011 44 1223 703 444 Asia Pacific: 61 425 792 927
--------------	-------------------------------------------------------------------------------------------------------------------------------------

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

Chapter 1 – Introduction	1
What's in the guide	1
Where to find additional information.....	1
Chapter 2 – RPM DAL Architecture.....	3
General DAL information	3
RPM architecture layers	5
Presentation layer.....	5
Service layer.....	5
Business object.....	5
Controller	6
Data access objects (DAO)	6
Component descriptions and standards	6
Chapter 3 – Detailed porting example.....	9
RPM DAL mechanisms	9
Properties file	9
DAL classes and interfaces	10
Chapter 4 – Required Merchandising System tables & columns for RPM 10.1	21

Chapter 1 – Introduction

This porting guide provides useful information to those who might be porting the Data Abstraction Layer (DAL) of the Retek Price Management (RPM) application. The reference implementation in this guide is Retek Merchandising System, version 10.1.

What's in the guide

The major components of the operations guide include:

Chapter 2 - DAL Architecture – This chapter discusses RPM's DAL design pattern in general.

Chapter 3 - Detailed DAL porting example – In this chapter, a sample class from the RPM DAL layer is examined in detail, and the specific actions on how to port this piece of the DAL to a different merchandising system are presented.

Chapter 4 - Required Merchandising System Tables & Columns for RPM 10.1 – This chapter contains a table that shows all of the RMS tables/columns used by RPM 10.1. It then shows which ones are required, in some form, no matter what merchandising system is used, for RPM 10.1 to function correctly.

Where to find additional information

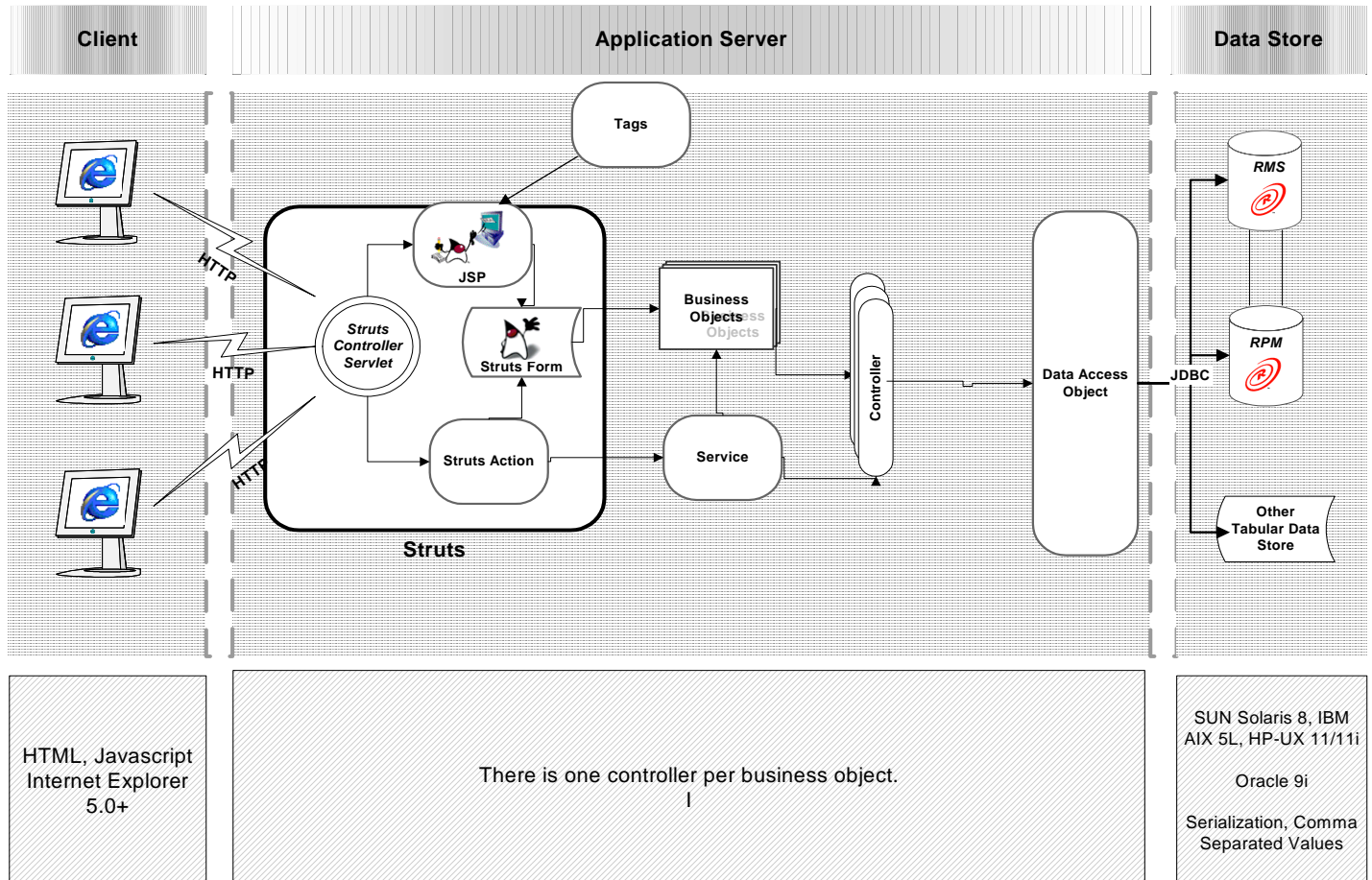
- RPM 10.1 Online Help
- RPM 10.1 User Guide
- RPM 10.1 Operations Guide – This guide, in particular, contains information useful to porting. It contains the details of the RMS tables and columns used by RPM.

Chapter 2 – RPM DAL Architecture

This chapter describes the RPM DAL architecture in general. In Chapter 3, we will look at a specific example of porting a piece of the DAL to a new merchandising system.

General DAL information

The chapter begins with a diagram of the architecture on the next page.



RPM 10.1 architecture

In the previous diagram notice that all Business Objects and services access the layer labeled Controller. The Controllers are the topmost portion of the DAL layer and are in the only layer touched by classes not part of the DAL implementation. The Controller classes do not need to be touched when porting RPM to a new persistence model/merchandising system.

The Controllers represent the logical means for gaining access to Business Objects. The Controllers invoke the lower levels of the DAL to obtain the actual objects. The DAO classes may need to be updated when moving to a new merchandising system.

Each DAO within RPM is defined by a Java interface that defines the methods that must be present upon any concrete DAO implementation. Concrete DAO implementations must implement their respective DAO interface. The interfaces do not need to be modified while porting.

At runtime, the Controller determines which DAO implementation to use by consulting the properties file, `com\retек\rpm\dao\conf\DAOType.properties`. This file is described in Chapter 3.

The following is the list of DAO interfaces that potentially need to have new implementations when moving to a new merchandising system. There are currently 24 such DAO interfaces. These are the DAO implementations that currently extract data from the RMS 10.1 system of Retek. Other DAO's within RPM work directly with RPM's own database tables and do not need to be ported when moving RPM to a new merchandising system.

- `com.retek.rpm.dao.DepartmentDAO`
- `com.retek.rpm.dao.HierarchyClassDAO`
- `com.retek.rpm.dao.SnapshotAlbumCatalogDAO`
- `com.retek.rpm.dao.PriceZoneGroupDAO`
- `com.retek.rpm.dao.PriceZoneDAO`
- `com.retek.rpm.dao.CompetitorDAO`
- `com.retek.rpm.dao.CompetitorStoreDAO`
- `com.retek.rpm.dao.CompetitorStoreItemLinkDAO`
- `com.retek.rpm.dao.ItemDAO`
- `com.retek.rpm.dao.ItemZoneInfoDAO`
- `com.retek.rpm.dao.UnitConvertorDAO`
- `com.retek.rpm.dao.SeasonsAndPhasesDAO`
- `com.retek.rpm.dao.UDADAO`
- `com.retek.rpm.dao.PendingCostChangeDAO`
- `com.retek.rpm.dao.PendingPriceChangeDAO`
- `com.retek.rpm.dao.PendingPromotionDAO`
- `com.retek.rpm.dao.ClearanceDAO`
- `com.retek.rpm.dao.WarehouseDAO`
- `com.retek.rpm.dao.SecurityDAO`

- `com.retek.rpm.dao.RMSSystemOptionsDAO`
- `com.retek.rpm.dao.PriceSuspendDAO`
- `com.retek.rpm.dao.ClearanceSuspendDAO`
- `com.retek.rpm.dao.BatchDAO`
- `com.retek.rpm.dao.CurrencyCodeDAO`

In Chapter 3, we examine the steps involved in porting the `HierarchyClassDAO` implementation. In the RPM Operations Guide, a detailed list of all of these DAO's will be given including information such as which tables they access, and which data is required for RPM to function.

RPM architecture layers

The following provides a brief description of the pieces of the entire RPM application.

Presentation layer

RPM 10.1 is built upon the Model-View-Controller (MVC) design pattern. The MVC pattern allows RPM to decouple the user interface from the business logic. RPM's implementation of the MVC pattern uses JavaServer Pages to provide the 'view', Java Servlets to provide the 'control', and Java classes to provide the business logic.

To assist in the implementation of the MVC pattern, RPM uses the Struts Framework. Struts is part of the Jakarta Project, sponsored by the Apache Software Foundation. The official Struts home page is at <http://jakarta.apache.org/struts>.

The major benefit of the MVC design pattern is the clearer separation of presentation from content. The design of the JSP that provides the user interface can be separated from most of the business logic that resides on the server. This separation of presentation from content offers a greater possibility for ease of maintenance, both the page that the user sees and the underlying logic.

Service layer

The Service layer provides the support for functional areas of RPM. Service objects have the responsibility for coordinating which Business Objects are needed by a presentation layer screen. It retrieves the necessary Business Objects processes data from those objects, and provides the information to the presentation layer. When you submit a request to the presentation layer, that request is handed to the appropriate service object where it is broken apart, and the necessary Business Objects are updated. The Service Layer also has the responsibility for coordinating transaction processing to ensure the ACID properties of the functional request. Finally, the Service Layer attempts to aggregate any validation problems detected so that you are presented with a list of problems that need to be fixed rather than have the problems reported one at a time.

Business object

A Business Object represents a specific concept in the business domain. For instance, within RPM, there are Business Objects for Department, Item and Pricing Strategies to name a few. The business object combines data about the business concept with the logic that processes that data.

Controller

Controllers provide the access point for retrieving specific Business Objects from their persistent store and updating the persistent form of the Business Object if it has changed. They also provide methods for creating new instances of Business Objects or for deleting Business Object instances that are no longer valid.

The controller is the interface to the Data Abstraction Layer. Business object code only uses the controller layer to obtain persistent data.

Data access objects (DAO)

Classes in the DAO layer abstract the actual persistence mechanism that is being used to persist Business Objects. Currently RPM persists its Business Objects in an Oracle database. The DAO layer provides the mechanism that allows RPM to be changed to a different persistence mechanism for representing the Business Objects. In those cases, only the DAO layer would need to be modified due to the change. The remainder of RPM would continue operating unchanged.

Component descriptions and standards

RPM is deployed using the technologies and versions described in this section.

Java Development Kit (JDK), version 1.3.1

Standard Java development tools from Sun Microsystems.

Java Server Pages (JSP), version 1.1

Allows Java and HTML to be combined within a web page. To the user this appears in the Web browser as a file with a .jsp extension. The JSP source is dynamically compiled into a servlet by the servlet container running in the web server. The servlet generates the necessary HTML content that the user sees.

Java Servlet, version 2.2

A servlet is a Java platform technology that allows a web application easier access to server side resources. The HTTP request from the client's browser is routed to the servlet, which then can process it as necessary and provide the appropriate response to the user.

Tomcat, version 3.2.3 (servlet container)

Used as the application server in the development environment.

Oracle9iAS Containers for J2EE (OC4J)

Used as the application server in the production environment.

Struts, version 1.0.2

An open source web development framework from the Jakarta Project and sponsored by the Apache Foundation. It provides three major components:

- A controller servlet that dispatches requests to appropriate RPM Action classes.
- JSP custom tag libraries, and associated support in the controller servlet, that supports RPM in providing an interactive form-based application.
- Utility classes to support XML parsing, automatic population of JavaBeans properties based on the Java reflection APIs, and internationalization of prompts and messages.

JDBC, version 1.0.2 (Java Database Connection)

Provides the support that allows RPM to submit SQL queries to the database and receive the result set for further processing. LOG4J.

LOG4J

An open source sub-project of the Jakarta Project. It provides a configurable framework for logging information gathered during the execution of an application.

JUnit Testing Framework

JUnit is an open source unit-testing framework provided by JUnit.org. This framework provides many tools and libraries that ease the task of developing comprehensive, automated unit tests for an application.

Chapter 3 – Detailed porting example

The DAL layer of RPM is designed to support the replacement of the persistence mechanism that stores the actual data used to create the Business Objects used in the upper layers of RPM.

In this chapter, we show an example of replacing the HierarchyClass DAO within RPM. The HierarchyClass DAO provides the data for the Business object that represents a particular class within the merchandising hierarchy. This example serves as a starting guide for the way to replace any general DAO object implementation within RPM.

In general, the following steps provide the easiest method of modifying the RPM DAO implementations:

1. Locate the particular DAO implementation you wish to replace.
2. Copy that implementation to your own java package.
3. Locate the SQL statements within the java code and modify them to suit your own merchandising system.
4. Compile the new classes and place the class files into your http server environment.
5. Modify the properties file `com\retek\rpm\dao\conf\DAOType.properties` to point to the new implementation.

RPM DAL mechanisms

Properties file

The properties file:

`com\retek\rpm\dao\conf\DAOType.properties`

contains the mapping from a key representing a type to the specific Data Access Object (DAO) class name that RPM should use to access the desired persistent data. As shipped, the contents of this file consist of a number of lines such as:

```
com.retek.rpm.dao.HierarchyClassDAO = com.retek.rpm.dao.db.HierarchyClassDbDAO
```

In the example above, the key type is `com.retek.rpm.dao.HierarchyClassDAO` and the specific class name is `com.retek.rpm.dao.db.HierarchyClassDbDAO`.

This information is used at runtime to determine the specific DAO implementation to use. For example, if a class such as:

```
com.ZZZ.rpm.dao.db.HierarchyClassDbDAO
```

where **ZZZ** is your companies name, were written to access a **ZZZ** specific merchandising system, then the `DAOType.properties` file entry for HierarchyClass would be modified to be:

```
com.retek.rpm.dao.HierarchyClassDAO = com.ZZZ.rpm.dao.db.HierarchyClassDbDAO
```

Any package structure for the class could be used, the previous is for illustrational purposes only.

DAL classes and interfaces

Within the java package `com.retek.rpm.dao` are located the RPM DAO interface classes. These all follow the naming convention `xxxDAO.java`. Each of these is a java interface that declares the methods that need to be provided by an implementation DAO class.

The *HierarchyClassDAO* interface looks like:

```
package com.retek.rpm.dao;

import com.retek.rpm.model.*;
import com.retek.rpm.exception.RpmException;
import java.util.Map;
import com.retek.rpm.model.key.*;

public interface HierarchyClassDAO extends DAO {
    public static final String COPYRIGHT = "Copyright 2002 Retek Inc.";
    public HierarchyClass load(ClassKey classKey, DepartmentKey deptKey)
        throws RpmException;
    public void save(HierarchyClass cls) throws RpmException;
    public Map getAllForDepartment(DepartmentKey deptKey)
        throws RpmException;
}
```

This interface does not need to be modified at all, but rather must be implemented by any concrete DAO implementation of *HierarchyClassDAO*. The shipped implementation of *HierarchyClassDAO* is *com.retek.rpm.dao.db.HierarchyClassDbDAO* and looks like:

```
package com.retek.rpm.dao.db;

import com.retek.rpm.model.*;
import com.retek.rpm.model.key.*;
import com.retek.rpm.util.DataAccessUtil;
import com.retek.rpm.dao.HierarchyClassDAO;
import com.retek.rpm.dao.db.oracle.rms.OracleClassHandler;
import com.retek.rpm.exception.RpmException;
import java.util.*;
import java.io.*;

public class HierarchyClassDbDAO implements HierarchyClassDAO {
    public static final String COPYRIGHT = "Copyright 2002 Retek Inc.";
```



```

public HierarchyClass load(ClassKey classKey, DepartmentKey deptKey)
                                throws RpmException {
    return OracleClassHandler.read(classKey, deptKey);
}

public void save(HierarchyClass cls) throws RpmException {
    throw new java.lang.UnsupportedOperationException();
} //end save

public Map getAllForDepartment(DepartmentKey deptKey) throws RpmException {
    return OracleClassHandler.readAllForDepartment(deptKey);
}
} //end class HierarchyClassDbDAO

```

The class *HierarchyClassDbDAO* does not do much on its own, but rather calls the helper class *com.retek.rpm.dao.db.oracle.rms.OracleClassHandler* to perform the actual database access. This pattern is followed throughout the DAO layer classes within RPM. This particular pattern is not mandatory, but an easy method of providing a new DAO implementation is to copy the existing DAO classes into a new package and modify them as you desire. You could also just modify the existing files in place without creating a new package.

Note that the method *save* in *HierarchyClassDbDAO* contains only the line :

```
throw new java.lang.UnsupportedOperationException();
```

This method is really only provided for testing purposes as RPM never creates classes within RMS. This is important because it means you do not have to implement this method either.

The complete code for the class *com.retek.rpm.dao.db.oracle.rms.OracleClassHandler* is as follows. For most instances of porting, only the SQL statements within the code need to be modified. In the following code, these statements are highlighted and will be examined afterwards.

```
package com.retek.rpm.dao.db.oracle.rms;
```

```
import java.sql.*;
```

```
import java.math.BigDecimal;
```

```
import java.util.Map;
```

```
import java.util.Hashtable;
```

```
import java.util.TreeMap;
```

```
import com.retek.rpm.util.*;
```

```
import com.retek.rpm.model.key.*;
```

12 Retek Price Management

```
import com.retek.rpm.model.HierarchyClass;
import com.retek.rpm.exception.DbOperationException;
import com.retek.rpm.dao.db.util.DbOperation;
import com.retek.rpm.common.UserContext;
import com.retek.rpm.common.User;

public class OracleClassHandler {

    public static final String COPYRIGHT = "Copyright 2002 Retek Inc.";

    public static HierarchyClass read(ClassKey classKey, DepartmentKey deptKey) throws DbOperationException {
        OracleClassReader reader = new OracleClassReader(classKey, deptKey);
        reader.perform();
        return reader.getResult();
    }

    public static Map readAllForDepartment(DepartmentKey deptKey) throws DbOperationException {
        OracleClassMultiReader multiReader = new OracleClassMultiReader(deptKey);
        multiReader.perform();
        return multiReader.getResult();
    }

    // Functional inner classes
    /**
     * This class reads a single Hierarchy class from the database
     */
    private static class OracleClassReader extends DbOperation {
        private ClassKey classKey;
        private DepartmentKey deptKey;
        private HierarchyClass cls;

        OracleClassReader(ClassKey classKey, DepartmentKey deptKey) {
            this.classKey = classKey;
            this.deptKey = deptKey;
            cls = null;
        }

        HierarchyClass getResult() {
            return cls;
        }

        protected void doPerform() throws Exception {
```

```

        buildClass();
    }

    private void buildClass() throws SQLException, DbOperationException {
        ResultSet rs;

        User user = UserContext.getUser();
        PreparedStatement stmt;

        if (user.isFullAccess()) {
            stmt = getPreparedStatement(CLASS_SQL);
        } else {
            stmt = getPreparedStatement(CLASS_SQL_SEC);
            stmt.setString(3, user.getName());
        }

        stmt.setBigDecimal(1, this.deptKey.getDecimalValue());
        stmt.setBigDecimal(2, this.classKey.getDecimalValue());

        rs = stmt.executeQuery();

        if (!rs.next()) {
            this.cls = null;
            return;
        }
        String name = rs.getString(1);

        this.cls = new HierarchyClass(new DepartmentClassCompositeKey(deptKey,
                                                                    classKey),
                                    name);

        if (rs.next()) {
            throw new DbOperationException("error.sql.baddata");
        }
    }

    private static final String CLASS_SQL
    = "SELECT class_name "
    + "FROM class WHERE class.dept = ? AND class.class = ?";

    private static final String CLASS_SQL_SEC
    = "SELECT "
    + " cl.class_name "

```

```

+ " FROM "
+ " CLASS cl "
+ " WHERE "
+ " (cl.dept = ? AND cl.class = ?) "
+ " AND "
+ " cl.dept IN "
+ " (SELECT sgpm.dept "
+ " FROM "
+ " SEC_USER_GROUP sug, "
+ " SEC_GROUP_PROD_MATRIX sgpm "
+ " WHERE "
+ " (sug.USER_ID = ? AND "
+ " sug.GROUP_ID = sgpm.GROUP_ID AND "
+ " cl.dept = sgpm.dept) "
+ " AND "
+ " ((cl.class = sgpm.class AND sgpm.UPDATE_IND = 'Y') "
+ " OR "
+ " (sgpm.class is NULL))) ";
}

/**
 * This class reads all Hierarchy classes for a Department from the database
 */
private static class OracleClassMultiReader extends DbOperation {
    private Map classes;
    private DepartmentKey deptKey;

    OracleClassMultiReader(DepartmentKey deptKey) {
        this.deptKey = deptKey;
        classes = new TreeMap();
    }

    Map getResult() throws DbOperationException {
        return classes;
    }

    protected void doPerform() throws Exception {
        buildClasses();
    }

    private static final String CLASSES_CACHE = "ClassesCache";

```

```

private static Hashtable getCache() {

    Hashtable classMap = (Hashtable)UserContext.getCache(CLASSES_CACHE);

    if (classMap == null) {
        classMap = new Hashtable(127);
        UserContext.setCache(CLASSES_CACHE, classMap);
    }

    return classMap;
}

private void buildClasses() throws SQLException, DbOperationException {
    ResultSet rs;
    User user = UserContext.getUser();
    PreparedStatement stmt;

    if (user.isFullAccess()) {
        stmt = getPreparedStatement(CLASS_SQL);
    } else {
        stmt = getPreparedStatement(CLASS_SQL_SEC);
        stmt.setString(2, user.getName());
    }

    stmt.setBigDecimal(1, this.deptKey.getDecimalValue());
    rs = stmt.executeQuery();

    Hashtable classCache = getCache();

    while (rs.next()) {
        BigDecimal classNum = rs.getBigDecimal(1);

        DepartmentClassCompositeKey dcck = new DepartmentClassCompositeKey(deptKey,
            ClassKey.create(classNum));

        HierarchyClass cls = (HierarchyClass)classCache.get(dcck);

        if (cls == null) {
            String name = rs.getString(2);

            cls = new HierarchyClass(dcck,
                name);

```

```

        classCache.put(dcck, cls);
    }
    classes.put(cls.getClassKey().toString(), cls);
}
}

private static final String CLASS_SQL
= "SELECT class, class_name FROM class "
+ "WHERE class.dept = ?";

private static final String CLASS_SQL_SEC
= "SELECT "
+ " cl.class, cl.class_name "
+ " FROM "
+ " CLASS cl "
+ " WHERE "
+ " (cl.dept = ?) "
+ " AND "
+ " cl.dept IN "
+ " (SELECT sgpm.dept "
+ " FROM "
+ " SEC_USER_GROUP sug, "
+ " SEC_GROUP_PROD_MATRIX sgpm "
+ " WHERE "
+ " (sug.USER_ID = ? AND "
+ " sug.GROUP_ID = sgpm.GROUP_ID AND "
+ " cl.dept = sgpm.dept) "
+ " AND "
+ " ((cl.class = sgpm.class AND sgpm.UPDATE_IND = 'Y') "
+ " OR "
+ " (sgpm.class is NULL))) ";
}
}

```

The *OracleClassHandler* class contains two private inner classes: *OracleClassReader* and *OracleClassMultiReader*. *OracleClassReader* is used to obtain a single row from the RMS CLASS table and return the data as a *HierarchyClass* object instance. *OracleClassMultiReader* is used to obtain all of the rows that belong to a given Department and return that data as a collection of *HierarchyClass* object instances. Both *OracleClassReader* and *OracleClassMultiReader* contain their own SQL for accessing the database. Both classes also extend the class *com.retek.rpm.dao.db.util.DbOperation*. *DbOperation* provides some helpful methods for using JDBC in a simplified fashion.

Next, we examine the class *OracleClassReader* in detail.

In the method *buildClass*, is the following code:

```
User user = UserContext.getUser();

PreparedStatement stmt;

if (user.isFullAccess()) {
    stmt = getPreparedStatement(CLASS_SQL);
} else {
    stmt = getPreparedStatement(CLASS_SQL_SEC);
    stmt.setString(3, user.getName());
}
```

In the above code, the current user is obtained from a call to the *UserContext* class. This user is then checked to determine if they are a **FullAccess** user. A **FullAccess** user is an RMS convention denoting a user who can access any data within the merchandising system. If the user does not have access to all data, then a more restrictive SQL statement that checks security tables within SQL is used to create the JDBC PreparedStatement. Both versions of the SQL statements will be examined shortly. If a new merchandising system does not have such security concepts, then the code could be simplified to:

```
stmt = getPreparedStatement(CLASS_SQL);
```

The method *getPreparedStatement* is a helper method inherited from the *DbOperation* superclass.

The String CLASS_SQL is:

```
"SELECT class_name FROM class WHERE class.dept = ? AND class.class = ?"
```

This statement obtains the `class_name` string from the RMS table `CLASS` where the `department` and `class` keys are equal to the provided values. In a JDBC prepared statement, the values that may be set are represented by ‘?’ in the SQL string. The code:

```
stmt.setBigDecimal(1, this.deptKey.getDecimalValue());
stmt.setBigDecimal(2, this.classKey.getDecimalValue());
```

sets the actual values in the *PreparedStatement* instance.

The statement is then executed and the value obtained in the following snippet:

[illegible]

```
name);
```

In order to modify this piece of code to obtain the information from your merchandising system, you need to modify the SQL statement to access the appropriate table(s) in your merchandising system. The table and column names are obvious candidates for modification.

In the RMS table CLASS, the class and dept columns are numeric values. If, in your corresponding table, the values happened to be strings, for example, you could modify the setter statements to look like:

```
stmt.setString(1, this.deptKey.toString());
        stmt.setString(2, this.classKey.toString());
```

The SQL string CLASS_SQL_SEC is more complex and looks into RMS security tables to determine if the current user has the authority to view the requested class. If you do not have corresponding security concepts, then you can just eliminate that portion of the code. If you do have such concepts, then here is the SQL string for CLASS_SQL_SEC and what it means.

```
"SELECT "
+ " cl.class_name "
+ " FROM "
+ " CLASS cl "
+ " WHERE "
+ " (cl.dept = ? AND cl.class = ?) "
+ " AND "
+ " cl.dept IN "
+ " (SELECT sgpm.dept "
+ " FROM "
+ " SEC_USER_GROUP sug, "
+ " SEC_GROUP_PROD_MATRIX sgpm "
+ " WHERE "
+ " (sug.USER_ID = ? AND "
+ " sug.GROUP_ID = sgpm.GROUP_ID AND "
+ " cl.dept = sgpm.dept) "
+ " AND "
+ " ((cl.class = sgpm.class AND sgpm.UPDATE_IND = 'Y') "
+ " OR "
+ " (sgpm.class is NULL))) ";
```

This SQL statement again takes the same parameters as the simpler SQL we previously discussed—the class and department keys. It also requires the current user name to look into the RMS table SEC_USER_GROUP. The SEC_USER_GROUP table is examined to determine if the current user is a member of an RMS group that has the department and class with update authority in the RMS table SEC_GROUP_PROD_MATRIX.

In the class *OracleClassMultiReader*, the general ideas for modification are the same. The SQL is slightly different. The CLASS_SQL string looks like:

```
"SELECT class, class_name FROM class "
```



```
+ "WHERE class.dept = ?";
```

Here, only the department key is used as a parameter. Thus, when the statement is executed, multiple rows may be returned in the result set. This is handled in the code:

```
rs = stmt.executeQuery();

Hashtable classCache = getCache();

while (rs.next()) {
    BigDecimal classNum = rs.getBigDecimal(1);

    DepartmentClassCompositeKey dcck = new DepartmentClassCompositeKey(deptKey,
        ClassKey.create(classNum));

    HierarchyClass cls = (HierarchyClass)classCache.get(dcck);

    if (cls == null) {
        String name = rs.getString(2);

        cls = new HierarchyClass(dcck,
            name);
        classCache.put(dcck, cls);
    }
    classes.put(cls.getClassKey().toString(), cls);
}
```

For each row returned, a *HierarchyClass* instance is created and placed within the collection to return. Note that first, the *classCache* hashtable is checked to see if this class has already been retrieved. If the instance is found within the cache, then the cached instance is used rather than constructing a new instance.

Again, the place to start modification is to change the SQL to match your merchandising system and to set the parameters according to the database types for your own system. The SQL string `CLASS_SQL_SEC` for the class *OracleClassMultiReader* again checks for user access to the classes before returning results.

Chapter 4 – Required Merchandising System tables & columns for RPM 10.1

This section defines which RMS tables and columns are currently used by RPM 10.1. It also shows which tables and columns are required for RPM to function correctly if you decide to use a different merchandising system. If you will be using a different merchandising system other than RMS 10.1, make sure that you have equivalent tables and columns available for RPM 10.1, which can be accessed through RPM's DAL layer. For a description of the RMS tables and columns, reference RMS 10.1's data model.

Note: X in the Required by RPM column means the given set of tables and columns are required.

RMS Table.Column	Required by RPM
CLASS.CLASS CLASS.CLASS_NAME CLASS.DEPT CLASS.VAT_IND	X
CLEAR_RESET_CALC.ITEM CLEAR_RESET_CALC.RESET_DOWNLOADED_DATE CLEAR_RESET_CALC.ZONE_GROUP_ID CLEAR_RESET_CALC.ZONE_ID	X
CLEAR_SUSP_DETAIL.ACTIVE_DATE CLEAR_SUSP_DETAIL.CLEARANCE CLEAR_SUSP_DETAIL.DOWNLOAD_DATE CLEAR_SUSP_DETAIL.ITEM CLEAR_SUSP_DETAIL.MARKDOWN_NBR CLEAR_SUSP_DETAIL.SELLING_UOM CLEAR_SUSP_DETAIL.UNIT_RETAIL CLEAR_SUSP_DETAIL.ZONE_GROUP_ID CLEAR_SUSP_DETAIL.ZONE_ID	X
CLEAR_SUSP_HEAD.APPROVAL_DATE CLEAR_SUSP_HEAD.APPROVAL_ID CLEAR_SUSP_HEAD.CLEARANCE CLEAR_SUSP_HEAD.CLEARANCE_DESC CLEAR_SUSP_HEAD.CREATE_DATE CLEAR_SUSP_HEAD.CREATE_ID CLEAR_SUSP_HEAD.REASON CLEAR_SUSP_HEAD.STATUS CLEAR_SUSP_HEAD.ZONE_GROUP_ID	X
CODE_DETAIL.CODE_DESC CODE_DETAIL.CODE_TYPE CODE_DETAIL.CODE	X

RMS Table.Column	Required by RPM
COMP_PRICE_HIST.COMP_RETAIL COMP_PRICE_HIST.COMP_RETAIL_TYPE COMP_PRICE_HIST.COMP_STORE COMP_PRICE_HIST.ITEM COMP_PRICE_HIST.MULTI_UNITS COMP_PRICE_HIST.MULTI_UNIT_RETAIL COMP_PRICE_HIST.REC_DATE COMP_PRICE_HIST.STORE	X
COMP_STORE.COMPETITOR COMP_STORE.CURRENCY_CODE COMP_STORE.STORE COMP_STORE.STORE_NAME	X
COMPETITOR.COMP_NAME COMPETITOR.COMPETITOR	X
COST_SUSP_SUP_DETAIL_LOC.COST_CHANGE COST_SUSP_SUP_DETAIL_LOC.ITEM COST_SUSP_SUP_DETAIL_LOC.LOC COST_SUSP_SUP_DETAIL_LOC.LOC_TYPE COST_SUSP_SUP_DETAIL_LOC.SUPPLIER COST_SUSP_SUP_DETAIL_LOC.ORIGIN_COUNTRY_ID	X
COST_SUSP_SUP_HEAD.COST_CHANGE COST_SUSP_SUP_HEAD.ACTIVE_DATE COST_SUSP_SUP_HEAD.STATUS	X
CURRENCIES.CURRENCY_CODE CURRENCIES.CURRENCY_DESC CURRENCIES.CURRENCY_COST_FMT CURRENCIES.CURRENCY_RTL_FMT CURRENCIES.CURRENCY_COST_DEC CURRENCIES.CURRENCY_RTL_DEC	X
DEPS.DEPT DEPS.DEPT_NAME DEPS.MARKUP_CALC_TYPE	X
DIFF_IDS.DIFF_ID DIFF_IDS.DIFF_TYPE	Required if you want differentiators for items. If these columns are not wanted, DAOs must be modified to remove these columns.

RMS Table.Column	Required by RPM
FUTURE_COST.PRICING_COST FUTURE_COST.ITEM FUTURE_COST.SUPPLIER FUTURE_COST.ORIGIN_COUNTRY_ID FUTURE_COST.LOCATION FUTURE_COST.LOC_TYPE FUTURE_COST.ACTIVE_DATE	X
IF_RPM_PRICE_EVENT.ITEM IF_RPM_PRICE_EVENT.STATUS IF_RPM_PRICE_EVENT.ZONE_GROUP_ID IF_RPM_PRICE_EVENT.ZONE_ID	X
IF_RPM_SMOOTHED_AVG.ITEM IF_RPM_SMOOTHED_AVG.STORE	X
ITEM_LOC.EOW_DATE ITEM_LOC.ITEM ITEM_LOC.LOC ITEM_LOC.LOC_TYPE ITEM_LOC.PRIMARY_CNTRY ITEM_LOC.PRIMARY_SUPP	X
ITEM_LOC_HIST.EOW_DATE ITEM_LOC_HIST.ITEM ITEM_LOC_HIST.LOC ITEM_LOC_HIST.LOC_TYPE ITEM_LOC_HIST.MONTH ITEM_LOC_HIST.SALES_ISSUES ITEM_LOC_HIST.SALES_TYPE	X
ITEM_LOC_SOH.ITEM ITEM_LOC_SOH.LOC ITEM_LOC_SOH.LOC_TYPE ITEM_LOC_SOH.STOCK_ON_HAND	X

RMS Table.Column	Required by RPM
ITEM_MASTER.CLASS ITEM_MASTER.DEPT ITEM_MASTER.FIRST_RECEIVED ITEM_MASTER.ITEM ITEM_MASTER.ITEM_DESC ITEM_MASTER.ITEM_LEVEL ITEM_MASTER.ITEM_PARENT ITEM_MASTER.LAST_RECEIVED ITEM_MASTER.ORIGINAL_RETAIL ITEM_MASTER.SELLABLE_IND ITEM_MASTER.STATUS ITEM_MASTER.SUB_CLASS ITEM_MASTER.TRAN_LEVEL	X
ITEM_MASTER.PACKAGE_SIZE ITEM_MASTER.PACKAGE_UOM	Required if you want to support multi-unit retails. If these columns are not wanted, DAOs must be modified to remove these columns
ITEM_MASTER.RETAIL_LABEL_TYPE ITEM_MASTER.RETAIL_LABEL_VALUE	Required if you want this additional reference information. If these columns are not wanted, DAOs must be modified to remove these columns
ITEM_MASTER.DIFF_1 ITEM_MASTER.DIFF_2 ITEM_MASTER.DIFF_3 ITEM_MASTER.DIFF_4 ITEM_MASTER.PARENT_DIFF_1 ITEM_MASTER.PARENT_DIFF_2 ITEM_MASTER.PARENT_DIFF_3 ITEM_MASTER.PARENT_DIFF_4	Required if you want differentiators for items. If these columns are not wanted, DAOs must be modified to remove these columns.
ITEM_SEASONS.ITEM ITEM_SEASONS.SEASON_ID ITEM_SEASONS.PHASE_ID	Required if you want Seasons and Phases. If these columns are not wanted, DAOs must be modified to remove these columns.
ITEM_SUPPLIER.ITEM ITEM_SUPPLIER.PRIMARY_SUPP_IND ITEM_SUPPLIER.SUPPLIER ITEM_SUPPLIER.VPN	X

RMS Table.Column	Required by RPM
ITEM_ZONE_PRICE.ITEM ITEM_ZONE_PRICE.MULTI_UNITS ITEM_ZONE_PRICE.MULTI_UNIT_RETAIL ITEM_ZONE_PRICE.MULTI_SELLING_UOM ITEM_ZONE_PRICE.SELLING_UOM ITEM_ZONE_PRICE.SELLING_UNIT_RETAIL ITEM_ZONE_PRICE.UNIT_RETAIL ITEM_ZONE_PRICE.ZONE_GROUP_ID ITEM_ZONE_PRICE.ZONE_ID ITEM_ZONE_PRICE.ITEM	X
ITEM_ZONE_PRICE.MKT_BASKET_CODE ITEM_ZONE_PRICE.LINK_CODE	<p>Required if you want to support Pricing Strategies that use Market Basket Codes.</p> <p>These are the Pricing Strategies that support Market Basket Codes or use link code functionality.</p> <ul style="list-style-type: none"> Competitive – Price by Code Margin Relationship – Competitor Target <p>If these columns are not wanted, DAOs must be modified to remove these columns.</p>
ORDHEAD.ORDER_NO ORDHEAD.STATUS	X
ORDLOC.ITEM ORDLOC.LOCATION ORDLOC.LOC_TYPE ORDLOC.ORDER_NO ORDLOC.QTY_ORDERED ORDLOC.QTY_RECEIVED ORDLOC.STATUS	X
PERIOD.CUR_454_YEAR PERIOD.CUR_454_MONTH PERIOD.VDATE	X
PHASES.PHASE_ID PHASES.PHASE_DESC	<p>Required if you want Phases for Seasons. If these columns are not wanted, DAOs must be modified to remove these columns.</p>

RMS Table.Column	Required by RPM
PRICE_EVENT_SKU.DETAIL_RECS_ATTEMPTED PRICE_EVENT_SKU.EVENT_TYPE PRICE_EVENT_SKU.ITEM PRICE_EVENT_SKU.PRICE_CHANGE	X
PRICE_SUSP_DETAIL.PRICE_CHANGE PRICE_SUSP_DETAIL.ITEM PRICE_SUSP_DETAIL.PC_SEQ_NO PRICE_SUSP_DETAIL.MULTI_UNITS PRICE_SUSP_DETAIL.MULTI_UNIT_RETAIL PRICE_SUSP_DETAIL.MULTI_SELLING_UOM PRICE_SUSP_DETAIL.PRICE_CHG_TYPE PRICE_SUSP_DETAIL.SELLING_UOM PRICE_SUSP_DETAIL.STATUS PRICE_SUSP_DETAIL.UNIT_RETAIL PRICE_SUSP_DETAIL.ZONE_GROUP_ID PRICE_SUSP_DETAIL.ZONE_ID	X
PRICE_SUSP_HEAD.ACTIVE_DATE PRICE_SUSP_HEAD.APPROVAL_DATE PRICE_SUSP_HEAD.APPROVAL_ID PRICE_SUSP_HEAD.CREATE_DATE PRICE_SUSP_HEAD.CREATE_ID PRICE_SUSP_HEAD.PC_ORIGIN PRICE_SUSP_HEAD.PC_TRAN_TYPE PRICE_SUSP_HEAD.PRICE_CHANGE PRICE_SUSP_HEAD.PRICE_CHANGE_DESC PRICE_SUSP_HEAD.REASON PRICE_SUSP_HEAD.STATUS PRICE_SUSP_HEAD.VENDOR_FUNDED_IND	X
PRICE_ZONE.CURRENCY_CODE PRICE_ZONE.DESCRPTION PRICE_ZONE.ZONE_GROUP_ID PRICE_ZONE.ZONE_ID	X
PRICE_ZONE_GROUP_STORE.PRIMARY_IND PRICE_ZONE_GROUP_STORE.STORE PRICE_ZONE_GROUP_STORE.ZONE_GROUP_ID PRICE_ZONE_GROUP_STORE.ZONE_ID	X
PROM_MIX_MATCH_BUY.ITEM PROM_MIX_MATCH_BUY.PROMOTION PROM_MIX_MATCH_BUY.DIFF_ID	X

RMS Table.Column	Required by RPM
PROM_MIX_MATCH_GET.ITEM PROM_MIX_MATCH_GET.PROMOTION PROM_MIX_MATCH_GET.DIFF_ID	X
PROM_THRESHOLD_SKU.ITEM PROM_THRESHOLD_SKU.PROMOTION PROM_THRESHOLD_SKU.DIFF_ID PROM_THRESHOLD_SKU.STATUS	X
PROMHEAD.PROMOTION PROMHEAD.STATUS	X
PROMSKU.ITEM PROMSKU.PROMOTION PROMSKU.DIFF_ID PROMSKU.STATUS	X
PROMSTORE.PROMOTION PROMSTORE.END_DATE PROMSTORE.START_DATE PROMSTORE.STORE	X
REPL_ITEM_LOC.ITEM REPL_ITEM_LOC.LOCATION REPL_ITEM_LOC.ACTIVATE_DATE REPL_ITEM_LOC.DEACTIVATE_DATE	X
SEC_GROUP_PROD_MATRIX.CLASS SEC_GROUP_PROD_MATRIX.DEPT SEC_GROUP_PROD_MATRIX.GROUP_ID SEC_GROUP_PROD_MATRIX.UPDATE_IND	DAOs can be changed to remove/change security requirement.
SEC_USER_GROUP.USER_ID SEC_USER_GROUP.GROUP_ID	DAOs can be changed to remove/change security requirement.
SEC_USER_PRIVS. SYSTEM_ACCESS_TYPE SEC_USER_PRIVS. USER_ID	DAOs can be changed to remove/change security requirement.
SEC_USER_PROD_MATRIX. COLUMN_CODE SEC_USER_PROD_MATRIX. USER_ID	DAOs can be changed to remove/change security requirement.
SEC_USER_ZONE_MATRIX. COLUMN_CODE SEC_USER_ZONE_MATRIX. USER_ID	DAOs can be changed to remove/change security requirement.
SEASONS.SEASON_ID SEASONS.END_DATE SEASONS.START_DATE	Required if you want to calculate Seasonal sell thru and to display dates in the season pop-up. If these columns are not wanted, DAOs must be modified to remove these columns.

RMS Table.Column	Required by RPM
SHIPMENT.RECEIVE_DATE SHIPMENT.SHIPMENT SHIPMENT.STATUS_CODE SHIPMENT.TO_LOC SHIPMENT.TO_LOC_TYPE SHIPSKU.ITEM SHIPSKU.SHIPMENT	X
SUBCLASS.CLASS SUBCLASS.DEPT SUBCLASS.SUBCLASS SUBCLASS.SUB_NAME	X
SYSTEM_OPTIONS.MULTI_PROM_IND	X
UDA.UDA_ID UDA.UDA_DESC UDA.DISPLAY_TYPE	Only required if you want user defined attributes for items. If these columns are not wanted, DAOs must be modified to remove these columns.
UDA_ITEM_DATE.ITEM UDA_ITEM_DATE.UDA_ID UDA_ITEM_DATE.UDA_DATE	Only required if you want user defined attributes for items. If these columns are not wanted, DAOs must be modified to remove these columns.
UDA_ITEM_FF.ITEM UDA_ITEM_FF.UDA_ID UDA_ITEM_FF.UDA_TEXT	Only required if you want user defined attributes for items. If these columns are not wanted, DAOs must be modified to remove these columns.
UDA_ITEM_LOV.ITEM UDA_ITEM_LOV.UDA_ID UDA_ITEM_LOV.UDA_VALUE	Only required if you want user defined attributes for items. If these columns are not wanted, DAOs must be modified to remove these columns.
UDA_VALUES.UDA_ID UDA_VALUES.UDA_VALUE UDA_VALUES.UDA_VALUE_DESC	Only required if you want user defined attributes for items. If these columns are not wanted, DAOs must be modified to remove these columns.
UNIT_OPTIONS.PRICE_PRIOR_CREATE_DAYS	X
UOM_CONVERSION.FACTOR UOM_CONVERSION.FROM_UOM UOM_CONVERSION.TO_UOM UOM_CONVERSION.OPERATOR	Only required if Package Size value is populated on ITEM_MASTER. . If these columns are not wanted, DAOs must be modified to remove these columns.
V_DIFF_ID_GROUP_TYPE.DIFF_TYPE V_DIFF_ID_GROUP_TYPE.ID_GROUP	X

RMS Table.Column	Required by RPM
VAT_ITEM.VAT_DATE VAT_ITEM.VAT_ITEM VAT_ITEM.VAT_RATE VAT_ITEM.VAT_REGION VAT_ITEM.VAT_TYPE	X
WH. WH WH. WH_NAME WH. STOCKHOLDING_IND WH. CURRENCY_CODE	X