

Oracle® Retail Strategic Store Solutions Implementation Guide

Oracle Retail Strategic Store Solutions to Merchandising
Operations Management Integration

Release 12.0

September 2007

Copyright © 2007, Oracle. All rights reserved.

Primary Author: Graham Fredrickson

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Value-Added Reseller (VAR) Language

(i) the software component known as **ACUMATE** developed and licensed by Lucent Technologies Inc. of Murray Hill, New Jersey, to Oracle and imbedded in the Oracle Retail Predictive Application Server - Enterprise Engine, Oracle Retail Category Management, Oracle Retail Item Planning, Oracle Retail Merchandise Financial Planning, Oracle Retail Advanced Inventory Planning and Oracle Retail Demand Forecasting applications.

(ii) the **MicroStrategy** Components developed and licensed by MicroStrategy Services Corporation (MicroStrategy) of McLean, Virginia to Oracle and imbedded in the MicroStrategy for Oracle Retail Data Warehouse and MicroStrategy for Oracle Retail Planning & Optimization applications.

(iii) the **SeeBeyond** component developed and licensed by Sun Microsystems, Inc. (Sun) of Santa Clara, California, to Oracle and imbedded in the Oracle Retail Integration Bus application.

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

(v) the software component known as **Crystal Enterprise Professional and/or Crystal Reports Professional** licensed by Business Objects Software Limited ("Business Objects") and imbedded in Oracle Retail Store Inventory Management.

(vi) the software component known as **Access Via**TM licensed by Access Via of Seattle, Washington, and imbedded in Oracle Retail Signs and Oracle Retail Labels and Tags.

(vii) the software component known as **Adobe Flex**TM licensed by Adobe Systems Incorporated of San Jose, California, and imbedded in Oracle Retail Promotion Planning & Optimization application.

(viii) the software component known as **Style Report**TM developed and licensed by InetSoft Technology Corp. of Piscataway, New Jersey, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

(ix) the software component known as **i-net Crystal-Clear**TM developed and licensed by I-NET Software Inc. of Berlin, Germany, to Oracle and imbedded in the Oracle Retail Central Office and Oracle Retail Back Office applications.

(x) the software component known as **WebLogic**TM developed and licensed by BEA Systems, Inc. of San Jose, California, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

(xi) the software component known as **DataBeacon**TM developed and licensed by Cognos Incorporated of Ottawa, Ontario, Canada, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

Contents

Preface	xiii
Audience.....	xiii
Related Documents	xiii
Customer Support	xiii
Review Patch Documentation	xiii
Oracle Retail Documentation on the Oracle Technology Network	xiv
Conventions	xiv
 1 Integration Overview	
Product Release Versions	1-1
Data Import from Oracle Retail Merchandising System and Oracle Retail Price Management	1-1
Generic Data Import Flow	1-3
Feed Methods.....	1-4
System Dependency.....	1-5
Oracle Retail Price Management to Oracle Retail Strategic Store Solutions Integration Overview	1-5
Oracle Retail Merchandising System to Oracle Retail Strategic Store Solutions Integration Overview	1-7
Oracle Retail Strategic Store Solutions to Oracle Retail Sales Audit Overview	1-9
Preconditions	1-10
System Flow Description	1-11
Existing Functionality Gaps	1-11
Oracle Retail Price Management.....	1-11
Oracle Retail Merchandising System	1-13
Discount Rule.....	1-15
Store Coupon	1-15
Data Import Field Width Maximums.....	1-16
 2 Integration Architecture	
Strategic Store Solutions to Oracle Retail Sales Audit Integration Architecture	2-1
RTLog Batch Generator	2-1
Sleep Interval	2-2
Maximum Transactions	2-2

Oracle Retail Sales Audit	2-2
Data Import.....	2-2
Error Handling	2-3
Import Status Logging.....	2-3
The Logic.....	2-4
Reprocessing a Bundle	2-4
Exception Flow	2-5
Logging.....	2-6
RTLog Mapping and Translation	2-7

3 Implementation Configuration

Data Import Spring Configurations	3-1
spring.properties	3-3
dimplogger.properties.....	3-4
Archive File Format.....	3-4
Oracle Retail Merchandising System Configuration	3-8
Oracle Retail Price Management Configuration	3-9
Data Requirements – Oracle Retail Strategic Store Solutions to Oracle Retail Sales Audit..	3-10

4 Capacity Planning

5 Customization Notes

Data Import Extension Points and Development	5-1
Import Adapter and Translator.....	5-3
SAXParserGenerator.....	5-3
Manually Editing Generated Code	5-3
Metadata	5-5
ImportControllerIfc.....	5-6
Strategic Store Solutions to Oracle Retail Sales Audit Extension Points and Development ...	5-6
Adding Data Elements to the RTLog Format	5-6
Creating a New Fixed Length Export Record Format	5-7
Exporting a Non-Fixed-Length Record Format.....	5-8
Object Factories.....	5-8
StoreServerConduit.xml.....	5-8
DomainObjectFactory.....	5-9
ExtractorObjectFactory	5-10
EntityMappingObjectFactory	5-10
RTLogMappingConfig.xml	5-10
RecordFormatObjectFactory	5-11
Configuration.....	5-11
The Store Server Conduit File	5-11
The Export Format Configuration file	5-12
The Entity Reader Configuration File	5-13
The Mapping Configuration File	5-13
Development and Testing Tools	5-14
Classes	5-14

Executables in the bin Directory 5-15

6 Known Issues and Troubleshooting

DepartmentDefaultTaxGroup..... 6-1

Character Restrictions for UOMs 6-1

POSlog 6-1

Preload Section of ItemImport 6-1

UTF-8..... 6-1

Third-party Tax and Employee Information 6-2

List of Figures

1-1	Integration Overview Including Strategic Store Solutions and Merchandising Operations Management Products 1-2	
1-2	Strategic Store Solutions to Oracle Retail Price Management Integration	1-6
1-3	Strategic Store Solutions and Oracle Retail Merchandising System Integration.....	1-8
1-4	High-Level Model for Oracle Retail Strategic Store Solutions-Oracle Retail Sales Audit Integration 1-10	
2-1	Data Import Tables Logical Data Mode.....	2-5
3-1	Adding Files To a Jar	3-6
3-2	Adding Files To A WinZip Archive	3-7
5-1	Employee Data Import Static Model.....	5-2

List of Tables

1-1	Functionality Gaps for Promotion Data Import.....	1-11
1-2	Functionality Gaps for Price Change Data Import	1-12
1-3	Functionality Gaps for Discount Rule Data Import.....	1-12
1-4	Functionality Gaps for Item Data Import.....	1-13
1-5	Functionality Gaps for Merchandise Hierarchy Data Import	1-14
1-6	Functionality Gaps for Store Hierarchy Data Import.....	1-15
1-7	Affected XML Elements	1-16
2-1	TransactionType (TRAT)	2-7
2-2	ReasonCode (REAC).....	2-8
2-3	OverrideReasonCodes (ORRC).....	2-9
2-4	ReturnReasonCodes (SARR)	2-10
2-5	SADT.....	2-10
2-6	TaxCode (TAXC).....	2-10
2-7	TenderTypes (TENT).....	2-10
2-8	TenderType ID (POS_TENDER_TYPE_HEAD).....	2-11
2-9	CCEM.....	2-12
2-10	Unit of Measure.....	2-12
2-11	Total ID for TOTAL type transactions	2-12
3-1	Spring Bean IDs Used For Each Of The Pluggable Components.....	3-1
3-2	Additional Spring Bean IDs Used For Each Of The Pluggable Components	3-2
3-3	Oracle Retail Merchandising System Default Values in the Back Office Item Maintenance Screen 3-8	
3-4	Oracle Retail Price Management Default Values	3-10
4-1	File Sizes	4-1
4-2	Bundle Size.....	4-2
4-3	Hard Drive Capacity	4-2
4-4	Item Import Data Volumes.....	4-2
5-1	Store Server Conduit File	5-9
5-2	EntityMappingObjectFactory Classes.....	5-10
5-3	RecordFormatObjectFactory Classes	5-11
5-4	Store Server Conduit File.....	5-11
5-5	Exportfile Utility Classes	5-15
5-6	bin Directory BAT Files.....	5-16

List of Examples

2-1	Sample JMX Configuration.....	2-6
2-2	Message Bean Definition.....	2-6
5-1	SAXParserGenerator utility command prompt.....	5-3
5-2	EmployeeAccessHandler Process DTO Before Children.....	5-4
5-3	EmployeeImportHandler Process DTO During Start.....	5-4
6-1	Tax File XML Schema Definition.....	6-2
6-2	Employee File XML Schema Definition.....	6-5

Preface

Audience

The Implementation Guide is intended for the Oracle Retail Point-of-Service integrators and implementation staff, as well as the retailer's IT personnel.

Related Documents

For more information, see the following documents:

- Oracle Retail Strategic Store Solutions Configuration Guide
- Oracle Retail Strategic Store Solutions Licensing Information
- Oracle Retail Strategic Store Solutions Relational Integrity Diagrams
- Oracle Retail Back Office documentation set
- Oracle Retail Labels and Tags documentation set
- Oracle Retail Central Office documentation set
- Oracle Retail Point-of-Service documentation set
- Oracle Retail Mobile Point-of-Service documentation set

Customer Support

- <https://metalink.oracle.com>

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

Review Patch Documentation

For a base release (".0" release, such as 12.0), Oracle Retail strongly recommends that you read all patch documentation before you begin installation procedures. Patch documentation can contain critical information related to the base release, based on new information and code changes that have been made since the base release.

Oracle Retail Documentation on the Oracle Technology Network

In addition to being packaged with each product release (on the base or patch level), all Oracle Retail documentation is available on the following Web site:

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

Documentation should be available on this Web site within a month after a product release. Note that documentation is always available with the packaged code on the release date.

Conventions

The following text conventions are used in this document:

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

Integration Overview

Product Release Versions

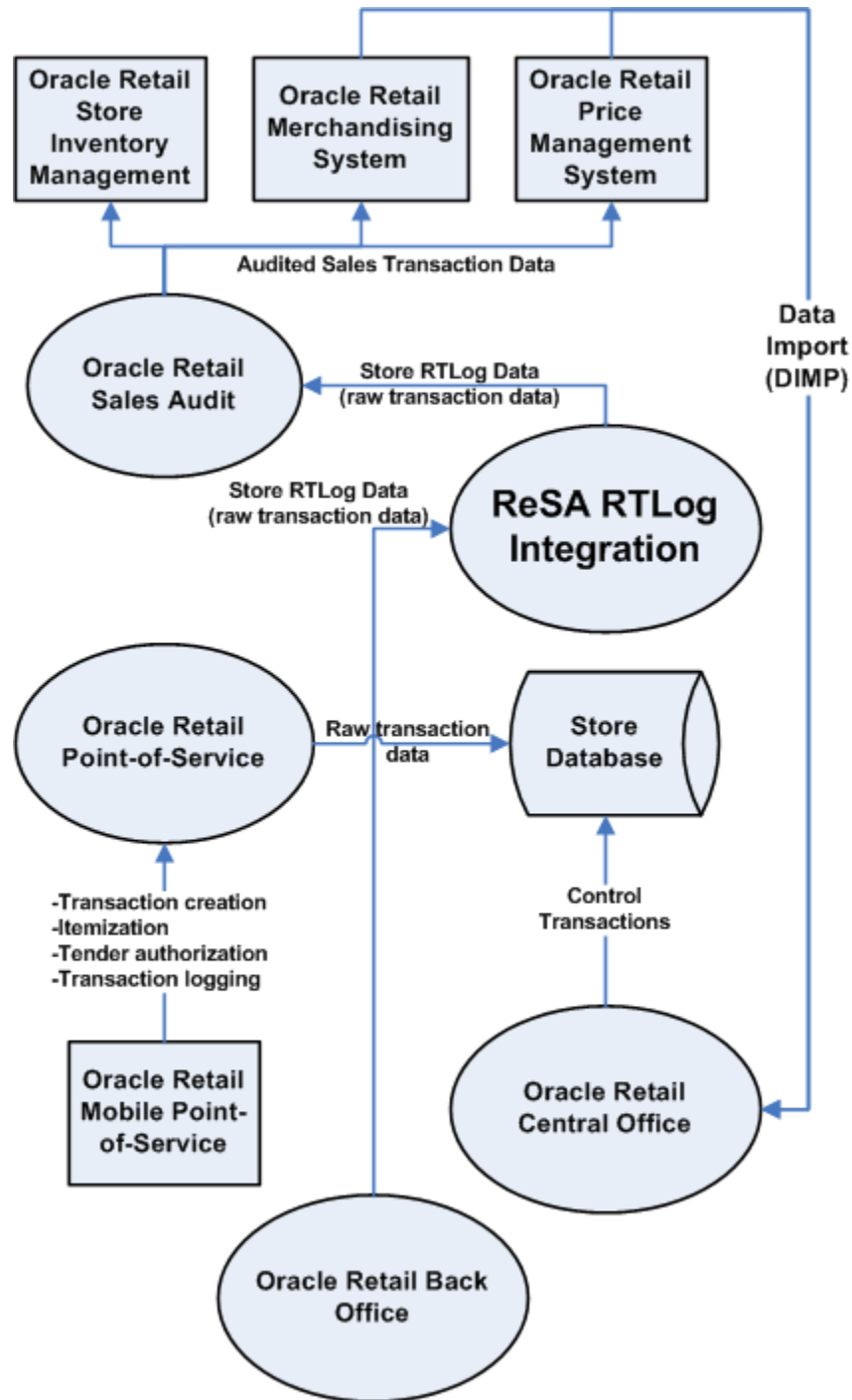
The following are the product release versions for this implementation:

- Oracle Retail Point-of-Service 12.0
- Oracle Retail Back Office 12.0
- Oracle Retail Central Office 12.0
- Oracle Retail Merchandising System 12.0.5
- Oracle Retail Price Management 12.0.5
- Oracle Retail Sales Audit 12.0.5

Data Import from Oracle Retail Merchandising System and Oracle Retail Price Management

The following is an overview diagram of an integration of Strategic Store Solutions and Merchandising Operations Management products, including a Data Import logical flow:

Figure 1–1 Integration Overview Including Strategic Store Solutions and Merchandising Operations Management Products



Seed data such as item, price and tax must be updated on an ongoing basis in the Store database (SDB) as well as Operational Data Store (ODS) to enable daily store operations. Typically the system of truth for such data is an enterprise system, such as Oracle Retail Merchandising System, Oracle Retail Price Management or a third-party product. The frequency and size of the data feeds varies from customer to customer. Imports are scheduled to be picked up by stores on a nightly basis. This interval is adjustable. See ["spring.properties"](#).

Note: DIMP is not the system of record for data correctness. All data coming into the data import module is assumed to be correct. This applies at two levels:

- First, the data must conform to the published XSDs.
- In addition, the database does not enforce referential integrity on the imported data, so the external system is responsible for not sending data that would create orphaned records in the database.

For example, there is no foreign key constraint enforced between the employee and store entities. A Kill And Fill import of the store hierarchy can result in a new set of stores that does not include a store for some existing employees. The external system that creates this import data must ensure that this type of situation does not occur.

Note: The base DIMP application supports parsing XML files only.

Generic Data Import Flow

The following describes the flow of a generic data import:

1. The flow begins with the Quartz Scheduler configured in Spring invoking the ImportIOAdapter of the DIMP Controller module.
2. The DIMP Controller picks up the import bundle, which is a compressed archive, and invokes the DIMP Translator.
3. The XML files are processed as input streams in order by DIMP translators: one for each import type.
4. The implementation of the ImportTranslatorIfc (as configured by Spring) retrieves an instance of an ImportControllerIfc from Spring and creates a new ImportBatch.
5. The translator begins to parse its document and calls initializeImport onto the controller.
6. The translator sets the batch size based upon its configuration.
7. The translator then loops through the elements in the document, creating a Data Transfer Object (DTO) for each complex element. The entity DTOs are processed one at a time in the order they are placed into the ImportBatch, with all Delete DTOs processing first, all Add DTOs second, then all Update DTOs last.
8. The controller retrieves an instance of the specified Data Access Object (DAO) from Spring based upon the key passed to it and calls initializeImport() on the DAO.
9. The translator then loops through the elements in the document, creating a Data Transfer Object (DTO) as each complex element. The entity DTOs are processed one at a time by placing them into the batch.
10. Each batch is processed as a transaction. Any data errors roll back that transaction. The import proceeds with the next batch.
11. The translator gives the ImportController a signal to process the batch after adding each DTO by calling processBatch().
12. If the batch size has been reached, the controller sends the batch to the DAO to be persisted.

13. The ImportDAOIfc loops through each DTO and delegates its data operation to a subordinate DAO.
14. Once the document parsing is complete, the translator notifies the controller, which processes the batch if there are any DTOs left over.
15. Finally, the controller calls completeImport() on the DAO, giving it the opportunity to copy data from temporary to production tables and drop temporary tables in case of a Kill And Fill, or release JDBC resources, and so forth.

Note: If you choose to retain any existing Oracle Retail Back Office or Oracle Retail Point-of-Service item-related functionality that creates or changes data types that are imported from Oracle Retail Merchandising System or any third party merchandising systems, you are responsible for handling and addressing any data overwrites performed by the import process.

Feed Methods

There are three feed methods:

Kill And Fill

Temporary tables are created at the beginning of a file's processing. Batches are written to the temporary tables. If the entire file is processed without error (all batches), the temporary table data replaces the production data and the temporary tables are dropped. If an error occurs, it is logged and the entire file import is aborted.

Full Incremental

Full Incremental is a fill type that performs adds, updates, or deletes expecting that all data attributes for a particular record are included in the file. Any missing attributes are provided default values.

Note: All columns for a row must be present in the import data.

For Full Incremental imports, each import XML data element must include all values. If some values are omitted from the import file, then the Data Import still updates the records in question, but uses default values for the omitted elements or attributes. Usually the default value chosen is **null**, **zero** or **false** unless otherwise specified in the XSD.

Delta Incremental

Delta Incremental is a fill type that produces dynamic update statements that allow for only those data attributes which are included in the file to be updated, leaving existing data attributes intact.

Note: Only those fields being updated are required in the import data.

System Dependency

Because some foreign key constraints are not being used, some dependencies might not actually exist. However, the logical dependencies are as follows:

- Tax depends on nothing
- Store Hierarchy/Stores depends on Tax (GeoCode)
- Employee depends on Store Hierarchy/Stores
- Merchandise Hierarchy depends on nothing
- Item depends on Tax and Merchandise Hierarchy
- Pricing depends on Item

Oracle Retail Price Management to Oracle Retail Strategic Store Solutions Integration Overview

Oracle Retail Price Management is a strategy-based pricing solution that suggests and assists with pricing decisions, yielding a more predictable and profitable outcome. Oracle Retail Price Management evaluates prices within a broad business context with real-time access to the following:

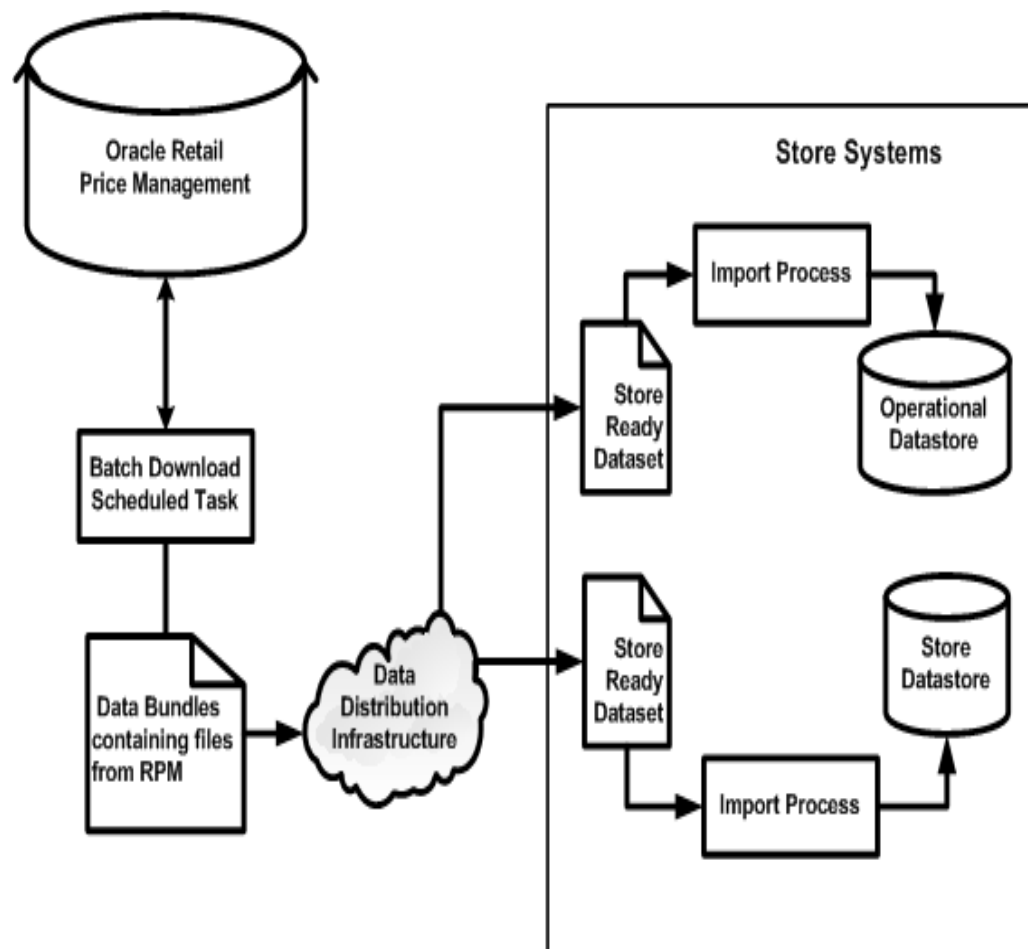
- Competitive and market data
- Projected sales impact
- Margin
- Pricing-based costs
- Current and projected inventory positions
- Markdown budgets

Oracle Retail Price Management provides a well-defined and efficient price change process that allows for aggregated permanent and clearance price change execution. Oracle Retail Price Management enables retailers to automate and streamline pricing strategies across the organization. Oracle Retail Price Management provides decision support to all pricing-focused business information to validate and approve pricing and markdown suggestions.

Note: This integration is one-way only. Oracle Retail Strategic Store Solutions changes are not communicated back up to Oracle Retail Price Management.

The following figure shows a high level overview of the integration.

Figure 1–2 Strategic Store Solutions to Oracle Retail Price Management Integration



Oracle Retail Merchandising System to Oracle Retail Strategic Store Solutions Integration Overview

Oracle Retail Merchandising System provides for core merchandising activities, including inventory replenishment, purchasing, and vendor management, in a global environment, across multiple retail channels. The solution incorporates three functional areas:

- Business foundation management
- Merchandise management
- Merchandise financial tracking

These functional areas enable retailers to streamline their business systems and unify business practices across their organization.

Oracle Retail Merchandising System is the main application for item, item location, merchandise hierarchy, stores and store (organizational) hierarchy data. This data is necessary for store operations and must be updated in the stores on an ongoing basis. Further, this data, particularly item data, can range in size from small incremental updates to large batch loads. The frequency and size of data feeds varies widely from customer to customer.

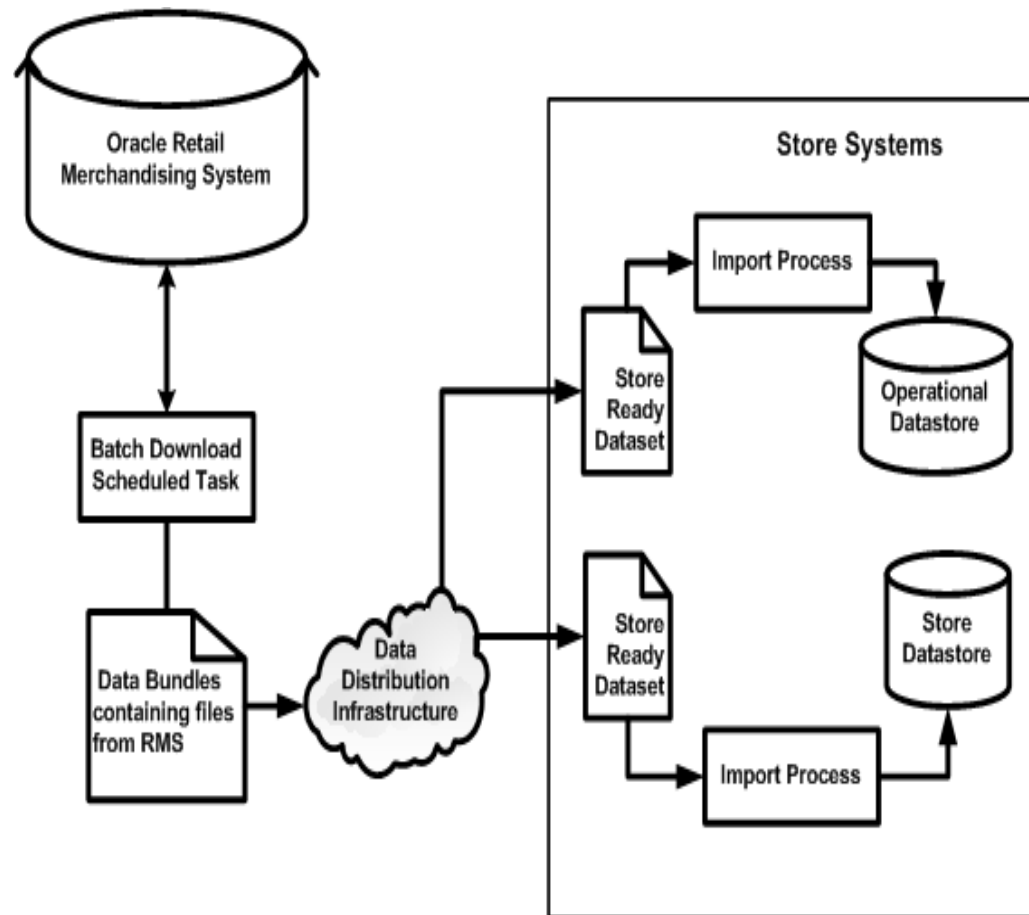
Note: This integration is one-way only. Oracle Retail Strategic Store Solutions changes are not communicated back up to Oracle Retail Merchandising System.

Note: There are some conditions required on data in order to filter out the Oracle Retail Merchandising System data being extracted to the XML files. This is required mainly because Oracle Retail Point-of-Service has these limitations on data types. Some of these conditions are:

- Store value length is less than or equal to 5
 - Chain value length is less than or equal to 4
 - Item value length is less than or equal to 14
 - UOM length is less than or equal to 2
 - Diff_1 length is less than or equal to 20
 - Diff_2 length is less than or equal to 6
 - Unit retail is less than or equal to 999999.99
-
-

The following figure shows a high level overview of the integration.

Figure 1–3 Strategic Store Solutions and Oracle Retail Merchandising System Integration



Oracle Retail Strategic Store Solutions to Oracle Retail Sales Audit Overview

The integration of the Oracle Retail Strategic Store Solutions products with the Oracle Retail Sales Audit (ReSA) application involves the following components:

Oracle Retail Strategic Store Solutions

The Oracle Retail Strategic Store Solutions logical component is comprised of Oracle Retail Point-of-Service, Back Office, and Central Office. RTLog data is created from Point-of-Service.

Oracle Retail Strategic Store Solutions RTLog Files

The RTLog file is the communication mechanism for providing data from the Oracle Retail Strategic Store Solutions to Oracle Retail Sales Audit. The RTLog is a transaction log file that is formatted specifically for Oracle Retail Sales Audit. Raw transaction data in the RTLog file is meant to update other Merchandise Operations Management applications, and is populated from Oracle Retail Strategic Store Solutions. The file is written to the physical file system by Oracle Retail Strategic Store Solutions for consumption by the transportation middleware.

Oracle Retail Strategic Store Solutions is responsible for writing the RTLog files to a configurable physical directory on the Store Server.

Note: RTLog files are encrypted. See *Oracle® Retail Merchandising System Operations Guide - Batch Overviews and Designs - Volume 1 Release 12.0.5*.

Integration Middleware

The integration middleware is a component that is responsible for polling the RTLog file produced by the Oracle Retail Strategic Store Solutions. This component has the following responsibilities:

- Polling the physical file system at a specified directory.
- Writing the RTLog file to a location that Oracle Retail Sales Audit expects.
- Cleaning and archiving the RTLog file once Oracle Retail Sales Audit has consumed the RTLog file.
- Error notification if the RTLog file is not able to be extracted successfully from a physical directory.

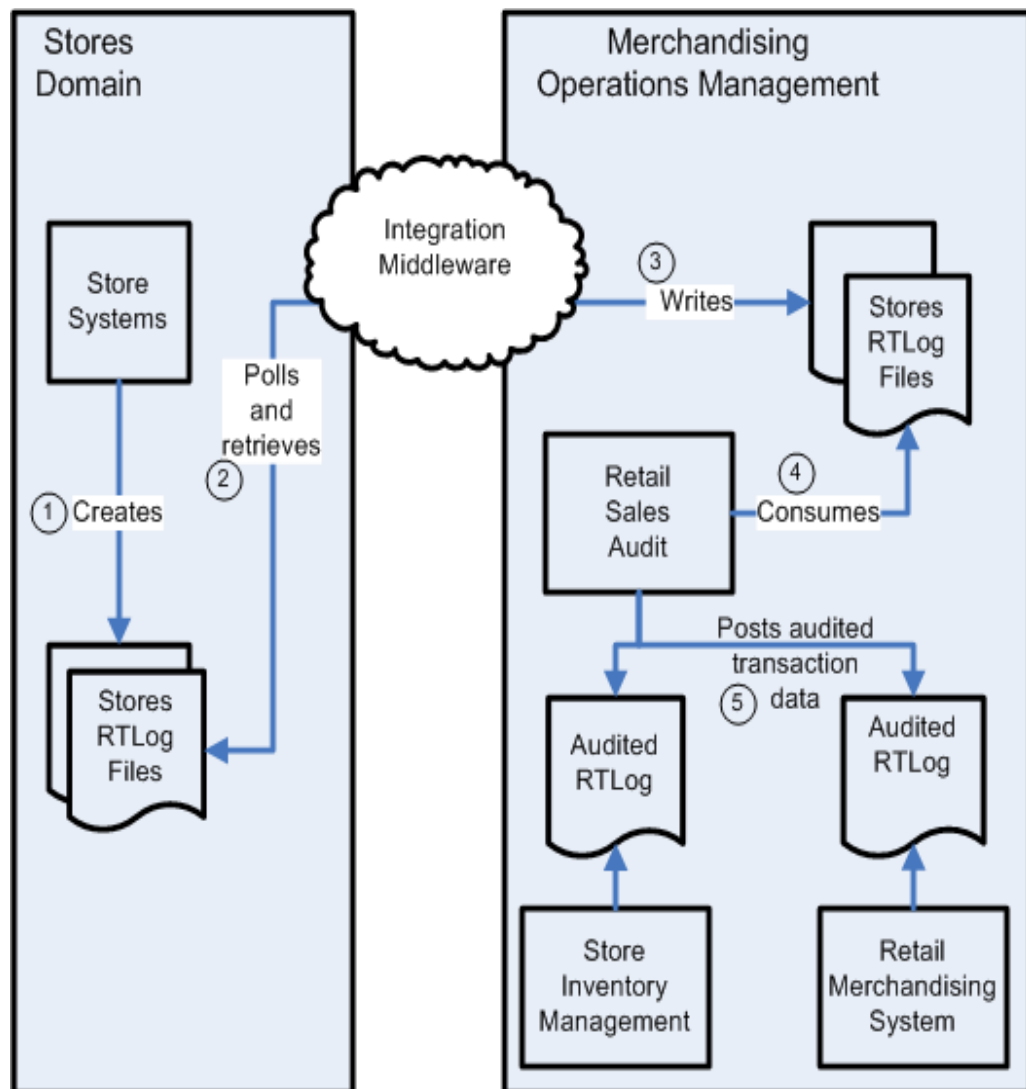
Note: The integration middleware is provided by the implementation team.

Oracle Retail Sales Audit

Oracle Retail Sales Audit is the gateway for transaction data updates to merchandising and inventory systems. The Oracle Retail Sales Audit consumes the RTLog file written to a specific directory by the integration middleware. Oracle Retail Sales Audit also sends audited data files to other Merchandise Operations Management applications for consumption.

The following figure depicts the two domains that are involved when integrating transaction data within the Oracle Retail suite.

Figure 1–4 High-Level Model for Oracle Retail Strategic Store Solutions-Oracle Retail Sales Audit Integration



Preconditions

The following preconditions must be observed for the system flow to function correctly:

1. Transport middleware requires read and write access to the physical file system to which Oracle Retail Strategic Store Solutions writes the RTLog file.
2. Transport middleware requires read and write access to the physical file system from which Oracle Retail Sales Audit reads the RTLog files.
3. Oracle Retail Strategic Store Solutions requires access to a physical file system to produce the RTLog file.

System Flow Description

The Point-of-Service client application generates transaction data and sends the transaction object structure to the Point-of-Service store server. The Point-of-Service store server populates the JDBC statement type and commits the transaction data to the store database. The Point-of-Service store server also populates the RTLog structure with the appropriate data extracted from the transaction object tree. The time increment at which data is sent to Oracle Retail Sales Audit is dictated by the retailer. If the RTLog is not successfully created due to unsupported mappings, the transaction identifier and exceptional condition is logged in detail on the Point-of-Service store server.

The overall flow shown in [Figure 1–4](#) is summarized in the following sequence:

1. Oracle Retail Strategic Store Solutions creates RTLog files.
2. Transport middleware scans directory that Oracle Retail Strategic Store Solutions writes the RTLog file to and reads in unprocessed RTLog files.
3. Transport middleware moves the RTLog file from the physical directory written to by Oracle Retail Strategic Store Solutions to a physical directory on an enterprise server defined by Oracle Retail Sales Audit.
4. Oracle Retail Sales Audit consumes the RTLog file written to a pre-defined directory by the transport middleware and executes data cleansing operations to produce audited transaction data.
5. Oracle Retail Sales Audit outputs audited RTLog-formatted transaction batch files and places the files into directories accessible by Merchandise Operations Management applications.

Existing Functionality Gaps

For this release, there are certain functionality gaps that exist that are not remedied at this time. This section describes these functional gaps, and the suggested resolution.

Oracle Retail Price Management

[Table 1–1](#) is a list of functionality gaps that exist for the Promotion data import.

Table 1–1 Functionality Gaps for Promotion Data Import

Identified Functionality Gap	Suggested Resolution
Oracle Retail Price Management does not download a start time.	Assume a start time of 00:00:00.
Oracle Retail Price Management does not download an end time.	Assume an end time of 23:59:59.
Oracle Retail Price Management supports a larger field (Change Value - Number) than does Strategic Store Solutions. This field is the amount, either monetary or percent, to be used to change or replace the current selling price for a sale unit of an item. Could result in loss of data in case of a very large discount amount	Gap to remain unchanged for this release.

Table 1–1 Functionality Gaps for Promotion Data Import

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

[Table 1–2](#) is a list of functionality gaps that exist for the Price Change data import.

Table 1–2 Functionality Gaps for Price Change Data Import

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

[Table 1–3](#) is a list of functionality gaps that exist for the Discount Rule data import.

Table 1–3 Functionality Gaps for Discount Rule Data Import

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

Table 1–3 Functionality Gaps for Discount Rule Data Import

Identified Functionality Gap	Suggested Resolution
Oracle Retail Price Management supports larger values and more precision than stores. Meaning of value (% , \$, or new price) is defined by Change Type.	Field length remains the same in Oracle Retail Price Management and Strategic Store Solutions.
Oracle Retail Price Management does not download a start time.	Assume a start time of 00:00:00.
Oracle Retail Price Management does not download an end time.	Assume an end time of 23:59:59.
Oracle Retail Price Management does not support threshold or limit.	Assume no threshold
Oracle Retail Price Management does not support the Number of Times per transaction field.	Assume -1, which means no limit to the number of times the promotion can be applied to a transaction.
Oracle Retail Price Management does not support the Accounting Method field.	Assume the discount.
Oracle Retail Price Management does not directly support the Allow Source to Repeat field.	Allow source to repeat.
Oracle Retail Price Management does not directly support the Deal Distribution field.	Assume target only.
Target Quantity field is not supported in Oracle Retail Price Management.	Assume target quantity of 1.

Oracle Retail Merchandising System

[Table 1–4](#) is a list of functionality gaps that exist for the Item import.

Table 1–4 Functionality Gaps for Item Data Import

Strategic Store Solutions Attribute	Identified Functionality Gap	Suggested Resolution
Cost	Cost data is not included in the Point-of-Service download file, but Oracle Retail Merchandising System has this data. However, Point-of-Service does not access item cost data from manufacturer.	Gap to remain unchanged for this release.
Sign/Label	This is not maintained by Oracle Retail Merchandising System.	Gap to remain unchanged for this release.
Manufacturer	Not included in the Point-of-Service download, but Oracle Retail Merchandising System has this data.	This value is null.
Planogram	Not maintained by Oracle Retail Merchandising System. Oracle Retail Merchandising System has a generic attribute that could be used for this purpose.	Gap to remain unchanged for this release.
Serialized	Not maintained by Oracle Retail Merchandising System. Point-of-Service uses this to prompt for serial number during order pickup.	Default to false for Oracle Retail Merchandising System imports.
Restocking Fee	Not maintained by Oracle Retail Merchandising System. Point-of-Service uses this to prompt for a restocking fee during returns.	Default to false for Oracle Retail Merchandising System imports.

Table 1–4 Functionality Gaps for Item Data Import

Strategic Store Solutions Attribute	Identified Functionality Gap	Suggested Resolution
Activation Required	Not maintained by Oracle Retail Merchandising System.	No attribute in Oracle Retail Merchandising System. Not used by Point-of-Service.
Registry Eligible	Not maintained by Oracle Retail Merchandising System.	No attribute in Oracle Retail Merchandising System. Not used by Point-of-Service.
Special Order Eligible	Prevents certain items from being placed on a special order. Not maintained by Oracle Retail Merchandising System.	Default to false for Oracle Retail Merchandising System imports.
Employee Discount Eligible	Identifies an item as eligible for an employee discount. Not maintained by Oracle Retail Merchandising System.	Default to true for Oracle Retail Merchandising System imports.
Damage Discount Eligible	Identifies an item as eligible for damage discount. Not maintained by Oracle Retail Merchandising System.	Default to true for Oracle Retail Merchandising System imports.
Size Entry Required	Not maintained by Oracle Retail Merchandising System. Point-of-Service uses this attribute during a sale or return to prompt for item size.	Default to false for Oracle Retail Merchandising System imports.
Itemizing	Strategic Store Solutions assumes item data is interpreted as local time. File creation has the local Oracle Retail Merchandising System time, but no timezone info.	Assume all Timestamps are relative to GMT.
Localization	Oracle Retail Merchandising System data file does not contain localized data for a store.	Accepts one localized text from Oracle Retail Merchandising System and use as all three: stores, user, customer.

Table 1–5 is a list of functionality gaps that exist for the Merchandise Hierarchy import.

Table 1–5 Functionality Gaps for Merchandise Hierarchy Data Import

Strategic Store Solutions Attribute	Identified Functionality Gap	Suggested Resolution
Merchant ID	Oracle Retail Merchandising System does not specify a merchant ID with any of the merchandise classification records sent with the Merchandise Hierarchy download.	Gap to remain unchanged for this release.

Table 1–6 is a list of functionality gaps that exist for the Store Hierarchy import.

Table 1–6 Functionality Gaps for Store Hierarchy Data Import

Strategic Store Solutions Attributes	Identified Functionality Gap	Suggested Resolution
Store Class	Strategic Store Solutions does not accept class.	Gap to remain unchanged for this release.
Store Class Description	Strategic Store Solutions does not accept class description.	Gap to remain unchanged for this release.
Store Format	Strategic Store Solutions does not accept format as part of the data import.	Gap to remain unchanged for this release.
Format Name	Store does not accept format name as part of the data import.	Gap to remain unchanged for this release.

Discount Rule

A discount rule that allows the customer to choose from a group of items is not functioning correctly in an Oracle Retail environment that is integrated with Oracle Retail Merchandising System.

Scenario: A promotion for "Buy X Get %/\$ off Y," where X or Y (or both) is from a group of items. For example, buy any pair of jeans get 10% off a t-shirt. The expectation for this discount is that the customer purchases ONE item from the jeans (department) to get 10% off ONE t-shirt from the t-shirt (department).

Oracle Retail Point-of-Service Behavior: Oracle Retail Point-of-Service interprets the "any pair of jeans" as "all pairs of jeans" and the "any t-shirt" as "all t-shirts" before the rule is invoked and the discount is applied to the transaction.

Issue: The current integration does not support the concept of a coupon at the class or department level. The integration communicates the department or class as the individual items in that group. When the store system receives all the coupon records it converts them to a buy one of each versus buy one from a list of items.

Fix: A fix is in development for Oracle Retail Point-of-Service and Oracle Retail Retail Merchandising System.

Store Coupon

Scenario: A store coupon for \$10 off the purchase of a single pair of jeans is applied to merchandise class that has 8 different pair of jeans. The expectation is that the coupon is applied against any one pair of jeans and the shopper would save \$10 of the purchase price.

Oracle Retail Point-of-Service Behavior: Oracle Retail Point-of-Service expects the shopper to purchase 8 different pair of jeans to receive the \$10 savings.

Issue: Oracle Retail Price Management supports the concept of a coupon at the class or department level but the integration approach is not working correctly. Oracle Retail Price Management sends down a coupon record for every item in the class. When the store system receives all the coupon records it converts them to a "buy one of each" versus "buy one from a list of items."

Fix: A fix is in development for Oracle Retail Point-of-Service and Oracle Retail Retail Price Management.

Data Import Field Width Maximums

Some fields can potentially overflow at the database level because the fields are not specifically limited in length by the Data Import XSDs. The following table lists the XML elements that are affected.

Table 1–7 Affected XML Elements

Import	Elements	Maximum Column Size
Employee Import	Employee > EmployeeFullName	VARCHAR(150)
	Employee > EmployeeLastName	VARCHAR(50)
	Employee > EmployeeFirstName	VARCHAR(50)
	Employee > EmployeeMiddleName	VARCHAR(50)
Item Import	Item > RetailStoreItem > POSIdentity @SupplierID	VARCHAR(20)
Merchandise Hierarchy Import	PreloadData > MerchandiseGroup > Description	VARCHAR(255)
	PreloadData > POSDepartment > POSDepartmentID	VARCHAR(14)
	PreloadData > POSDepartment > ParentPOSDepartmentID	VARCHAR(14)
	HierarchyList > Hierarchy@Name	VARCHAR(14)
	HierarchyList > Hierarchy > LevelList > Level@Name	VARCHAR(120)
	HierarchyList > Hierarchy > NodeList > Node@ParentNodeID	VARCHAR(14)
	ierarchyList > Hierarchy > NodeList > Node@ID	VARCHAR(14)
Pricing Import	PricingImport > PriceChange @ID	VARCHAR(20)
	PricingImport > PriceChange > Item @ID	VARCHAR(14)
	PricingImport > PriceChange > Item @TemplateType	VARCHAR(8)
	PricingImport > PriceChange @TemplateType	VARCHAR(8)
	PricingImport > PricePromotion @ID	VARCHAR(20)
	PricingImport > PricePromotion @TemplateType	VARCHAR(8)
	PricingImport > PricePromotion @TemplateType	VARCHAR(8)
	DiscountRule > Sources > Source @ID	VARCHAR(14)
	DiscountRule > Targets > Target @ID	VARCHAR(14)
	DiscountRule > Sources > Source @ID	VARCHAR(14)
	DiscountRule > Sources > Source @ID	VARCHAR(10)

Table 1–7 (Cont.) Affected XML Elements

Import	Elements	Maximum Column Size
Store Hierarchy Import	PreloadData > StoreRegion > RegionID	VARCHAR(14)
	PreloadData > StoreRegion > RegionName	VARCHAR(120)
	PreloadData > StoreDistrict > DistrictID	VARCHAR(14)
	PreloadData > StoreDistrict > RegionID	VARCHAR(14)
	PreloadData > RetailStore > GeoCode	VARCHAR(10)
	PreloadData > StoreDistrict > DistrictName	VARCHAR(120)
	PreloadData > RetailStore > LocationName	VARCHAR(150)
	PreloadData > RetailStore > DistrictID	VARCHAR(14)
	PreloadData > RetailStore > RegionID	VARCHAR(14)
	PreloadData > RetailStore > GeoCode	VARCHAR(10)
	PreloadData > RetailStore > Address > AddressLine1	VARCHAR(30)
	PreloadData > RetailStore > Address > AddressLine2	VARCHAR(30)
	PreloadData > RetailStore > Address > AddressLine3	VARCHAR(30)
	PreloadData > RetailStore > Address > City	VARCHAR(30)
	PreloadData > RetailStore > Address > State	VARCHAR(30)
	PreloadData > RetailStore > Address > PostalCode	VARCHAR(30)
	PreloadData > RetailStore > Address > Territory	VARCHAR(30)
	PreloadData > RetailStore > Address > Country	VARCHAR(30)
	PreloadData > RetailStore > Address > TelephoneCountryCode	VARCHAR(30)
	PreloadData > RetailStore > Address > TelephoneAreaCode	VARCHAR(30)
	PreloadData > RetailStore > Address > TelephoneLocalNumber	VARCHAR(30)
	HierarchyList > Hierarchy@Name	VARCHAR(120)
	HierarchyList > Hierarchy > LevelList > Level@Name	VARCHAR(120)
	HierarchyList > Hierarchy > NodeList > Node@Name	VARCHAR(120)
	HierarchyList > Hierarchy > NodeList > Node@Description	VARCHAR(255)

Table 1–7 (Cont.) Affected XML Elements

Import	Elements	Maximum Column Size
Tax Import	GEOCode > GeoCodeID	VARCHAR(10)
	GEOCode > TaxJurisdictionName	VARCHAR(50)
	GEOTaxJurisdiction > GeoCodeID	VARCHAR(10)
	TaxAuthority > TaxAuthorityName	VARCHAR(40)
	TaxAuthority > GeoCodeID	VARCHAR(10)
	TaxableGroup > TaxGroupName	VARCHAR(120)
	TaxableGroup > TaxGroupDescription	VARCHAR(255)
	TaxAuthority > AddressLine	VARCHAR(30)
	TaxAuthority > City	VARCHAR(30)
	TaxAuthority > State	VARCHAR(30)
	TaxAuthority > PostalCode	VARCHAR(30)
	TaxAuthority > CountryCode	VARCHAR(30)
	TaxGroupRule > TaxTypeName	VARCHAR(30)
	TaxGroupRule > TaxRuleName	VARCHAR(40)
	TaxGroupRule > TaxRuleDescription	VARCHAR(255)

Integration Architecture

Strategic Store Solutions to Oracle Retail Sales Audit Integration Architecture

The Point-of-Service terminal is the platform that the Point-of-Service client application resides on. The cashier and the store manager interact with the Point-of-Service client application, which generates transaction data. The Point-of-Service client application sends a serialized object structure representing the sales transaction to the Point-of-Service store server residing on the In-Store-Processor (ISP). The ISP is responsible for logging the raw transaction data to the store database.

The major components of the Strategic Store Solutions to Oracle Retail Sales Audit integration are:

- **RTLog Export Daemon Technician**

Processes configuration settings from the Store Sever Conduit XML file; settings include sleep interval, maximum number of transactions per batch, export directory name, object factory class names, and export configuration files names.

Starts the RTLog Export Daemon Thread.

- **RTLog Export Daemon Thread**

Starts the export process on a periodic basis based on the configured sleep interval. Calls the RTLog Batch Generator.

- **RTLog Batch Generator**

Creates a list of transactions ready for export and calls the Export File Generator.

- **Export File Generator**

Reads the transactions in the list and formats the export data based on the export configuration files.

In this integration, the Point-of-Service store server also maps the transaction object structure to RTLog format and places the RTLog-formatted transaction into a file. The individual components that comprise the RTLog generation are described in the following subsections.

RTLog Batch Generator

The RTLog Batch Generator is a Java class that reads transactions from the store database and creates a physical RTLog file. The file format follows the standards outlined in *Oracle Retail Merchandising System Operations Guide - Batch Overviews and Designs - Volume 1 Release 12.0.5*.

The RTLog Batch Generator consumes a configuration file that has the settings outlined in the following sections.

Sleep Interval

The RTLog batch generator runs in a daemon mode, which periodically outputs RTLog files created by pulling transactions from the database. In this configuration, Oracle Retail Sales Audit processes one or more RTLog files from any given store.

Maximum Transactions

The Maximum Transactions setting puts a cap on the number of RTLog transactions read from the store database during a processing cycle. If the number of transaction available is less than the maximum transactions setting, the RTLog Batch Generator reads the number of transactions available.

If Maximum Transactions is set to **-1**, then there is no limit to the number of RTLog transactions.

Oracle Retail Sales Audit

The Oracle Retail Sales Audit system is responsible for sales audit functionality at the store and at the corporate level. Store operations make use of Oracle Retail Sales Audit's functionality to determine over/short situations in stores, and make the necessary adjustments to raw transaction data in order to ensure integrity of data being sent to backend merchandising and store inventory systems.

Oracle Retail Sales Audit consumes unaudited transaction data in RTLog batch format. It then subjects the transaction data to numerous checks, and indicates exceptional conditions leading to out-of-balance situations. The Oracle Retail Sales Audit system outputs cleansed or audited RTLog data to be consumed by Oracle Retail Merchandising System (RMS), Oracle Retail Price Management (RPM), and Oracle Retail Store Inventory Management (SIM).

Data Import

The Data Import (DIMP) utility is the application that enables the import of data from both Oracle Retail Merchandising System and Oracle Retail Price Management to Point-of-Service.

Note: When discussing Data Import, functionality applies to both Oracle Retail Merchandising System and Oracle Retail Price Management.

The Data Import (DIMP) subsystem and components are designed to enable external systems to send large volumes of data to the Oracle Retail Strategic Store Solutions applications. The primary intent of this functionality is to allow for initial data seeding or routine data loading (and optional purging) to occur for such types of data as:

- Taxation
- Merchandise Hierarchy
- Store Hierarchy
- Employee
- Item

- Pricing

Note: For ItemImport.xml, refer to the xsd. Some attributes are labeled **required**. All attributes listed as required in the xsd must be included in the ItemImport.xml file. See "[Archive File Format](#)" in Chapter 3 for more information about ItemImport.xml.

Note: When an item is imported without a POSDepartmentID, that particular item not associated with a POSDepartment. When the item is viewed in Back Office, the POSDepartment list defaults its selection to the first department in the list.

Note: Taxation and Employee information are not provided by Oracle Retail Merchandising System or Oracle Retail Price Management. The Taxation and Employee information comes from third-party systems.

For more information, see "Third-party Tax and Employee Information" in Chapter 6.

Error Handling

Strategic Store Solutions applications are not the system of record for data correctness. Error handling is limited to logging errors during the import and performing a retry in certain cases. Because the data imports can be interdependent, a failure in one file import results in an abort of the import of the rest of the files in the order.

There were no changes made to the base data model to support the new data import subsystem. However, a few tables have been added to take care of data import error handling and to support any recovery or retry mechanism that can be put place in future (that may be custom developed).

For the current implementation, all Kill And Fill imports are applied into temporary tables. Once the import of the complete bundle is successful, the data is written onto the main tables. If any data operation fails, the entire file import is aborted. A FAILURE status message is logged for each of those files.

Incremental file imports continue even if a data operation fails. In that case, the error is logged and it is the customer's responsibility to decide how to handle the failed operations.

The act of aborting the import is configurable and can be changed based on implementation requirements.

Import Status Logging

- In case of failure in opening the bundle or reading a file in the bundle, the status in the tables is MA_STS_BNDL_IMP - FAILED.

No other status is logged in any other table.

- In case of failure in parsing a file, the statuses are:
 - MA_STS_BNDL_IMP - PROCESSED

- MA_STS_FL_IMP - FAILED for that file and all other files that are dependent on that file.
- MA_FL_IMP_FLRS - Failure exception details of the file.
- In case of failure while persisting a batch:
 - If Kill and Fill then:

MA_STS_BNDL_IMP - PROCESSED

MA_STS_FL_IMP - FAILED for that file and all other files that are dependent on that file.

MA_FL_IMP_FLRS - Failure exception details of the file that has failed.

- If Full Incremental or Delta Incremental then:

MA_STS_BNDL_IMP - PROCESSED

MA_STS_FL_IMP - PARTIALLY PROCESSED for that file only.

MA_FL_IMP_FLRS - Failure exception details of the files that have failed.

The Logic

MA_STS_BNDL_IMP

This is the Bundle Import Status, which has the processing status at the bundle level. In a case where an input/output error occurs, such as unable to open the bundle or read a file from the bundle, the status is logged as FAILED. In all other cases where there is no input/output error, the status is PROCESSED. This is because a bundle can contain more than one file, and it is, from a performance standpoint, degenerative to keep track of how many files there are in the bundle and how many of them have succeeded and how many have failed. Therefore, unless an input/output error is encountered, the status PROCESSED is logged into the table.

MA_STS_FL_IMP

File Import Status maintains the processing status of each file in a bundle. The status FAILED for a file indicates that there is a parsing exception, or there is a failure while persisting a Kill And Fill file (as complete processing is aborted in case of Kill And Fill). If a failure is logged in this table for a file, then all other files in the bundle that are dependent on the failed file also have a FAILED status.

The status PARTIALLY PROCESSED for a Full Incremental or Delta Incremental import indicates there is a failure in persisting a batch. This status is irrespective of the number of records in the file. In an incremental type of import, a batch of records with no exceptions is persisted to the database and committed. Therefore, to note a FAILED status we must know how many records there are in the file, how many batches do these records form and the processing status of each of the batch. Performance wise this is not advisable.

Also, if a bundle is re-processed, a FAILED status on an incremental file causes the file to be processed again, generating more exceptions.

MA_FL_IMP_FLRS

Any failures encountered are logged in this table.

Reprocessing a Bundle

This facility is provided to reprocess any file that failed, that is, has a FAILED status in MA_STS_FL_IMP. No change is needed in the bundle to process a file again. If the same bundle is reprocessed, all the files with a status FAILED in MA_STS_FL_IMP are reprocessed. Therefore, if an incremental file has already crossed the point of parsing,

(an exception while persisting) then the status for that file must never be logged as FAILED, as some of the batches might have been persisted and reprocessing the file generates more errors.

Exception Flow

- If there is a failure in any insert operation for a file of the Kill And Fill variety, the exception is logged and the complete file is aborted. Import of any subsequent file in the sequence that depends upon the failed/aborted file is also aborted. This is done to ensure that partial data inserts from the file are not performed, compromising the integrity of the data in the database. Import of files that do not depend on this particular file is not impacted.
- If an operation (insert, update, delete) fails during the processing of an incremental file, delta or full, the current batch is aborted and subsequent batches are processed. The errors are logged for the failed batch and processing continues, starting with the next batch of the data in the file.

The figure below shows the logical data model for the tables being used in error handling in Data import.

Figure 2-1 Data Import Tables Logical Data Mode



The archive file status is logged as CONSISTENT or INCONSISTENT in the table ImportBundleStatus, with the BundleID of the archive.

If an exception is encountered during the import of a file, the record where the problem is encountered is logged in the table ImportRecordStatus.

The exception is then sent up to the Data Import Controller where a FAILED status is logged on to the table ImportFileStatus. If the import has been successful for a file, a status of SUCCESS is inserted in the table.

Instrumentation for application monitoring can be provided by exposing beans to JMX through Spring, which orchestrates the process of creating JMX management interfaces for beans, and removes the need to compile them to the JMX API.

The following example must be configured in the Spring PersistenceContext.xml file.

Example 2–1 Sample JMX Configuration

```
<bean id="mbeanServer"
class="org.springframework.jmx.support.MBeanServerFactoryBean"/>

<bean id="exporter" class="org.springframework.jmx.export.MBeanExporter">
  <property name="beans">
    <map>
      <entry key="bean:name=EmployeeImportDAOKey"
value-ref="EmployeeImportDAO"/>
    </map>
  </property>
  <property name="server" ref="mbeanServer"/>
</bean>

<bean id="EmployeeImportDAO"
class="com._
360commerce.commerceservices.employee.importdata.dao.EmployeeImportDAO"/>
```

Logging

At various points in the import process, exceptions such as SQLException and SAXException might be generated. They are generally rethrown as ImportExceptions and passed up the chain to the DIMP Controller, as well as logged for error tracking and resolution.

DIMP introduces a new Spring-based logging object to provide message consistency and allow retailer customization of messages. The underlying logging uses Apache Commons logging as the interface, and Log4j for the logging implementation. A MessageLogger is retrieved from the Spring service context. The logger gets message templates from a property file. Customers can define the layout of these messages to suit their needs, using the following format, where {x} is a placeholder for input data from the calling program:

```
Message from {0} with {1} information.
```

The Spring bean ID used for the pluggable message logger component is shown in [Table 3–1, "Spring Bean IDs Used For Each Of The Pluggable Components"](#). The mapping is shown below.

Example 2–2 Message Bean Definition

```
<bean id="service_MessageBuilder" class="com._
360commerce.commerceservices.importdata.MessageBuilder" singleton="true"
lazy-init="true">
```

```

    <property name="prefix"><value>${dimp.prefix}</value></property>
    <property name="texts">
      <list>
        <value>${dimp.text1}</value>
        <value>${dimp.text2}</value>
        <value>${dimp.text3}</value>
      </list>
    </property>
  </bean>

```

RTLog Mapping and Translation

The following tables identify the changes to the RTLog codemapping that enables Oracle Retail Sales Audit to consume RTLogs generated by Strategic Store Solutions applications.

For more RTLog information, see the *Oracle® Retail Merchandising System Operations Guide - Batch Overviews and Designs - Volume 1 Release 12.0.5*.

Note: New Oracle Retail Sales Audit code is highlighted with *italics*.

New Point-of-Service code is highlighted in **bold**.

Table 2–1 TransactionType (TRAT)

TransactionType	TRAT (Static)	TRAS (Static) Sub-Transaction Type
(1) Sale	SALE	<i>SALE</i>
(2) Return	RETURN	<i>RETURN</i>
(3) Void	PVOID	<i>VOID</i>
(4) NoSale	NOSALE	<i>NOSALE</i>
(1) Sale where an even exchange is made	EEXCH	<i>EXCH</i>
(6) OpenStore	OPEN	<i>OSTORE</i>
(7) CloseStore	DCLOSE	<i>DSTORE</i>
(8) OpenRegister	OPEN	<i>OREG</i>
(9) CloseRegister	CLOSE	<i>CRGRC</i>
(10) OpenTill	OPEN	<i>OTILL</i>
(11) CloseTill	CLOSE	<i>CTILL</i>
	TOTAL	<i>CTILLT</i>
(12) LoanTill	LOAN	<i>LOTILL</i>
(13) PickupTill	PULL	<i>PUTILL</i>
(14) SuspendTill	NOSALE	<i>STILL</i>
(15) ResumeTill	NOSALE	<i>RTILL</i>
(16) PayinTill	PAIDIN	<i>PITILL</i>
(17) PayoutTill	PAIDOU	<i>POTILL</i>
(18) HousePayment	PAIDIN	<i>HOUSE</i>
(19) LayawayInitiate	PAIDIN	<i>LAYINT</i>

Table 2–1 TransactionType (TRAT)

TransactionType	TRAT (Static)	TRAS (Static) Sub-Transaction Type
(20) LayawayComplete	SALE	LAYCMP
(21) LayawayPayment	PAIDIN	LAYPAY
(22) LayawayDelete	PAIDOU	LAYDEL
(23) OrderInitiate	PAIDIN	ORDINT
(24) OrderComplete	SALE	ORDCMP
(25) OrderCancel	PAIDOU	ORDCAN
(26) OrderPartial	SALE	ORDPAR
(27) BankDepositStore	NOSALE	BANK
(35) Instant Credit Enrollment	NOSALE	INSCRE
(36) Redeem	RETURN	REDEEM
(37) Enter Training Mode	NOSALE	NTRAIN
(38) Exit Training Mode	NOSALE	XTRAIN
(40) Payroll Payout	PAIDOU	PAYOUT
(41) Enter Transaction Reentry	NOSALE	NTRENT
(42) Exit Transaction Reentry	NOSALE	XTRENT
Any transaction where Transaction.TransactionStatusCode = (3) Canceled	VOID	CANCEL
Any transaction where Transaction.TrainingMode= 'ON	NOSALE	TRAIN
Any transaction where Transaction.TransactionStatusCode = (4) Suspend Transaction	NOSALE	SUSPND

Note: (4) Suspend Transactions get sent in the RTLog. Subsequent resume or cancel actions simply change the transaction status code to (6) Resume Transaction or (7) Canceled Suspended Transaction. A new transaction is not created, hence no subsequent RTLog record, except if the Suspended Transaction is Resumed then SOLD, upon which a SALE transaction is created.

Table 2–2 ReasonCode (REAC)

Reason entered by cashier for some transaction types. Required for Paid In and Paid out transaction types, but can also be used for voids, returns, and so forth.	REAC	Description
Till.TillPayInReasonCodes (53) BadCheckPayment	NSF	NSF Check Payment
Till.TillPayInReasonCodes(54) VendingMachineRevenue	TPIVMR	TillPayIn VendingMachineRevenue
Till.TillPayInReasonCodes(55) Miscellaneous	TPIMSC	TillPayIn Miscellaneous
Till.TillPayrollPayOutReasonCodes (1) PayrollAdvance	PAYRL	Payroll Payout
Till.TillPayrollPayOutReasonCodes (2) FinalPay	PAYRL	Payroll Payout

Table 2–2 ReasonCode (REAC)

Reason entered by cashier for some transaction types. Required for Paid In and Paid out transaction types, but can also be used for voids, returns, and so forth.	REAC	Description
Till.TillPayOutReasonCodes (56) Postage	<i>TPOP</i>	<i>TillPayOut Postage</i>
Till.TillPayOutReasonCodes (57) Supplies	<i>TPOS</i>	<i>TillPayOut Supplies</i>
Till.TillPayOutReasonCodes (58) Entertainment	<i>TPOE</i>	<i>TillPayOut Entertainment</i>
Sale.NoSaleReasonCodes(1) CustomerChange	<i>NSCC</i>	<i>NoSale CustomerChange</i>
Sale.NoSaleReasonCodes(2) ChangeForRegister	<i>NSCFR</i>	<i>NoSale ChangeForRegister</i>
Sale.PostVoidReasonCodes(1) IncorrectPrice	<i>PVIP</i>	<i>PostVoid IncorrectPrice</i>
Sale.PostVoidReasonCodes(2) DiscountIncorrect	<i>PVDI</i>	<i>PostVoid DiscountIncorrect</i>
Sale.PostVoidReasonCodes(3) CustomerChangedMind	<i>PVCCM</i>	<i>PostVoid CustomerChangedMind</i>
Sale.PostVoidReasonCodes(4) AssociateError	<i>PVAE</i>	<i>PostVoid AssociateError</i>
Sale.PostVoidReasonCodes(5) OtherFormPayment	<i>PVOFP</i>	<i>PostVoid OtherFormPayment</i>
Sale.PostVoidReasonCodes(6) Other	<i>PVO</i>	<i>PostVoid Other</i>
Where transaction type = (18) House Payment	<i>HOUSE</i>	<i>House Payment</i>
Where transaction type = (19) Layaway Initiate	<i>LAYINT</i>	<i>Layaway Initiate</i>
Where transaction type = (21) Layaway Payment	<i>LAYPAY</i>	<i>Layaway Payment</i>
Where transaction type = (23) Order Initiate	<i>ORDINT</i>	<i>Order Initiate</i>
Where transaction type = (22) Layaway Delete	<i>LAYDEL</i>	<i>Layaway Delete</i>
Where transaction type = (25) Order Cancel	<i>ORDCAN</i>	<i>Order Cancel</i>

Table 2–3 OverrideReasonCodes (ORRC)

Reason an item price is overridden at the Point-of-Service.	ORRC	Dynamic
(3) Defective	<i>D</i>	<i>Damaged Goods</i>
(5) SignageError	<i>S</i>	<i>Incorrect Signage</i>
(2) CompetitionPrice	<i>CP</i>	<i>CompetitionPrice</i>
(1) AdPrice	<i>AP</i>	<i>AdPrice</i>
(4) ManagersSpecial	<i>MS</i>	<i>ManagersSpecial</i>

Table 2–4 ReturnReasonCodes (SARR)

The reason an item is returned.	SARR	Dynamic
(33) Defective	01	Damaged
(33) Defective	02	Defective
(11) WrongColor	06	Color Not As Shown
(45) CustomerChangedMind	19	CustomerChangedMind
(55) PriceAdjustment	20	PriceAdjustment

Table 2–5 SADT

The type of discount within a promotion.	SADT	Dynamic
(2402,2006,2303,2105) Saturday Morning Special	SATSPL	Saturday Morning Special
(2410,2014,2311,2113) Senior Citizen	SENCIT	Senior Citizen
(2428,2022,2329,2121) Competition Special	CMPSPL	Competition Special
(2436,2030,2337,2139) Store Coupon	SCOUP	Store Coupon

Table 2–6 TaxCode (TAXC)

Tax code to represent whether it is a state tax type, provincial tax, and so forth.	TAXC	Dynamic
TOTTAX	TOTTAX	Aggregate total of tax excluding VAT

Table 2–7 TenderTypes (TENT)

High-level grouping of tender types.	TENT	Static
CASH Cash	CASH	Cash
CRDT Credit Card	CCARD	Credit Card
CHCK Check	CHECK	Personal Check
ECHK E-Check	CHECK	Personal Check
TRAV Travelers Check	CHECK	Personal Check
MBCK Mail Bank Check	CHECK	Personal Check
QPON Manufacturers Coupon	COUPON	Coupon
DBIT Debit Card	DCARD	Debit Card
MNYO Money Order	MORDER	Money Order
GCRD Gift Card	VOUCH	Voucher (gift cert. or credit)
GICT Gift Certificate	VOUCH	Voucher (gift cert. or credit)
STCR Store Credit	VOUCH	Voucher (gift cert. or credit)
MACT Mall Certificate	VOUCH	Voucher (gift cert. or credit)
PRCH Purchase Order	VOUCH	Voucher (gift cert. or credit)
VOUCH Voucher	VOUCH	Voucher (gift cert. or credit)

Table 2–8 TenderType ID (POS_TENDER_TYPE_HEAD)

Tender Type ID. Low level grouping of tender types.	POS_TENDER_TYPE_HEAD	Notes
CASH Cash	1000 CASH Cash - primary currency	
CHCK Check	2000 CHECK Personal Check	
TRAV Travelers Check	2020 CHECK Traveler Check	
QPON Manufacturers Coupon	5000 COUPON Manufacturers Coupons	
DBIT Debit Card	8000 DCARD Debit Card	
MNYO Money Order	6000 MORDER Money Orders	
GICT Gift Certificate	4030 VOUCH Gift Certificate	
GCRD Gift Card	4040 VOUCH Gift Card	
STCR Store Credit	4050 VOUCH Store Credit	
MACT Mall Certificate	4060 VOUCH Mall Certificate	
PRCH Purchase Order	4070 VOUCH Purchase Order	
VOUCH Voucher	4080 VOUCH PrePaid	Use for Orders and Layaways
ECHK E-Check	2030 CHECK E-Check	
MBCK Mail Bank Check	2040 CHECK Mail Bank Check	
Visa	3000 CCARD Visa	
MasterCard	3010 CCARD Mastercard	
AmEx	3020 CCARD American Express	
Discover	3030 CCARD Discover	
DinersClub	3040 CCARD Diners Club - N. America	
HouseCard	3120 CCARD House Card	
JCB	3130 CCARD JCB	
CASH Cash Alternate Currency	<i>1010 CASH Cash Alternate Currency</i>	
CHCK Check Alternate Currency	<i>2050 CHECK Personal Check Alternate Currency</i>	
TRAV Travelers Check Alternate Currency	<i>2060 CHECK Travelers Check Alternate Currency</i>	
STCR Store Credit Alternate Currency	<i>4090 VOUCH Store Credit Alternate Currency</i>	
GICT Gift Certificate	<i>4100 VOUCH Gift Certificate Alternate Currency</i>	

Table 2–9 CCEM

Credit card input type	CCEM	Dynamic
Manual	T	Terminal Used
MSR	MSR	Magnetic Strip Read

Table 2–10 Unit of Measure

Unit of Measure	
'LF' 'linear feet'	'LF' 'linear feet'
'LM' 'linear meters'	'LM' 'linear meters'
'PN' 'pounds net'	'LBS' 'POUNDS'
'KG' 'kilograms'	'KG' 'KILOGRAM'
'UN' 'units'	'EA' 'EACH'

Table 2–11 Total ID for TOTAL type transactions

Total ID (Reference Number 1) for TOTAL type transactions.	
1000 CASH Cash - primary currency	CASH
2000 CHECK Personal Check	CHCK
2020 CHECK Traveler Check	TRAVCHK
5000 COUPON Manufacturers Coupons	QPON
8000 DCARD Debit Card	DEBITCARD
6000 MORDER Money Orders	MNYORDER
4030 VOUCH Gift Certificate	GIFTCERT
4040 VOUCH Gift Card	GIFTCARD
4050 VOUCH Store Credit	STCREDIT
4060 VOUCH Mall Certificate	MALLCERT
4070 VOUCH Purchase Order	PRCHORDER
2030 CHECK E-Check	ECHECK
2040 CHECK Mail Bank Check	MBCHECK
3000 CCARD Visa	CCARDVisa
3010 CCARD Mastercard	CCARDMCard
3020 CCARD American Express	CCARDAmEx
3030 CCARD Discover	CCARDDisc
3040 CCARD Diners Club - N. America	CCARDDiner
3120 CCARD House Card	CCARDHCard
3130 CCARD JCB	CCARDJCB
1010 CASH Cash Alternate Currency	CASHAC
2050 CHECK Personal Check Alternate Currency	PCHECKAC
2060 CHECK Alternate Currency	TCHECKAC
4090 VOUCH Store Credit Alternate Currency	STCRDTAC
4100 VOUCH Gift Certificate Alternate Currency	GIFTCERTAC

Implementation Configuration

Data Import Spring Configurations

The system has been designed to support a pluggable model. The DIMP Controller, ImportTranslator, ImportController, ImportDAO, MessageLogger and scheduler are all designed to be configurable at deployment time. This is accomplished through the use of Spring as a deployment framework. Each of these classes is only accessed through their interface. Therefore, any new implementations only need to support the interfaces to be used by the subsystem. Introducing an alternate implementation is done through updates to the Spring configuration file. No additional code changes are necessary.

Table 3–1 includes the set of Spring bean IDs used for each of the pluggable components.

Table 3–1 Spring Bean IDs Used For Each Of The Pluggable Components

Spring bean ID	Provided implementation	Additional configuration
service_ MerchandiseHierarchyImportTranslator	com._ 360commerce.commerceservices.item.hierarchy.importdata.MerchandiseHierarchyImportTranslator	batchSize=1000
service_ StoreHierarchyImportTranslator	com._ 360commerce.commerceservices.store.hierarchy.importdata.StoreHierarchyImportTranslator	batchSize=1000
service_TaxImportTranslator	com._ 360commerce.commerceservices.tax.importdata.TaxImportTranslator	batchSize=1000
service_ EmployeeImportTranslator	com._ 360commerce.commerceservices.employee.importdata.EmployeeImportTranslator	batchSize=1000
service_MessageLogger	com._ 360commerce.commerceservices.common.importdata.MessageLogger	messages=messageSource

Table 3–1 Spring Bean IDs Used For Each Of The Pluggable Components

Spring bean ID	Provided implementation	Additional configuration
service_ImportJobTrigger	org.springframework.scheduling.quartz.SimpleTriggerBean	JobDetail=service_ImportOrchestratorStartDelay=10000RepeatInterval=10000
service_ImportOrchestrator	org.springframework.scheduling.quartz.JobDetailBean	com._360commerce.commerceservices.importdata.ImportOrchestratorJob
DIMP_Scheduler	org.springframework.scheduling.quartz.SchedulerFactoryBean	triggers=service_ImportJobTriggerAutoStartup=trueApplicationContextSchedulerContextKey=applicationContextWaitForJobsToCompleteOnShutdown=true

These setting can be found in the `ServiceContext.xml` file packaged in the `config.jar` under the `/config/context` package.

The `web.xml` in `WEB-INF` directory has the following configuration under the `web-app` section.

```
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/schedulingContext-quartz.xml</param-value>
</context-param>
```

The following servlet should also be configured to start up automatically. The servlet loads the context configuration files. Because the `schedulingContext-quartz.xml` file is configured in the context, this file is loaded by the servlet. `SchedulerFactoryBean` is configured to start on load; hence it is invoked and starts the scheduler timer.

```
<servlet>
<servlet-name>context</servlet-name>
<servlet-class>org.springframework.web.context.ContextLoaderServlet</servlet-class>
>
<load-on-startup>1</load-on-startup>
</servlet>
```

[Table 3–2](#) includes additional sets of Spring bean IDs used for each of the pluggable components.

Table 3–2 Additional Spring Bean IDs Used For Each Of The Pluggable Components

Spring bean ID	Provided implementation	Additional configuration
persistence_ImportController	com._360commerce.commerceservices.importdata.ImportController	batchSize=1000
persistence_MerchandiseHierarchyImportDAO	com._360commerce.commerceservices.item.hierarchy.importdata.dao.MerchandiseHierarchyImportDAO	dataSource=persistence_dataSource

Table 3–2 Additional Spring Bean IDs Used For Each Of The Pluggable Components

Spring bean ID	Provided implementation	Additional configuration
persistence_ StoreHierarchyImportDAOTarget	com._ 360commerce.commerceservices.store. hierarchy.importdata.dao.StoreHierarc hyImportDAO	dataSource=persistence_dataSource
persistence_TaxImportDAOTarget	com._ 360commerce.commerceservices.tax.i mportdata.dao.TaxImportDAO	dataSource=persistence_dataSource
persistence_ EmployeeImportDAOTarget	com._ 360commerce.commerceservices.empl oyee.importdata.dao.EmployeeImport DAO	dataSource=persistence_dataSource

These settings can be found in the PersistenceContext.xml file packaged in the config.jar under the /config/context package.

By default, the ImportController's batch size is set to 1000 and all the translators are also using the same. However, each individual translator can be configured separately to optimize the import per the size of the data operation. Spring sets the batch size value onto the translator when instantiated. It is the responsibility of the translator to call setBatchSize(int) with that value onto the ImportController.

Notice that the ID of the DAO beans end with Target. This is because the ID that is actually used by the application returns a Proxy Bean configured to intercept method calls to the DAO and associate transactions with them. Upon ImportExceptions thrown by those methods, the transaction is rolled back.

Several configuration files exist containing settings specific to DIMP. Properties are read when the server starts, so any changes require a server restart before they take effect.

spring.properties

```
#####
## Global settings (applicable to OC4J and WAS) ##
#####

# directory in which incoming data import bundles arrive
importdata.file.path=C:/temp/dataimport/incoming

# directory in which dimp bundles are archived after processing
importdata.archive.path=C:/temp/dataimport/archive

# true/false whether data import scheduler should scan importdata.file.path
execute.import=false

# the delay in milliseconds to start the import and price change triggers (900000
= 15 minutes)
import.scheduler.startdelay=60000
pricechange.scheduler.startdelay=120000

# the delay between import and price change scheduler executions (86400000 = 24
hours)
import.scheduler.repeatinterval=86400000
pricechange.scheduler.repeatinterval=3600000

# name of the DIMP logger config file
```

```
logger.filename=dimplogger
```

`importdata.file.path` and `importdata.archive.path` are file-system dependent. Windows systems would use paths such as:

```
C:/temp/dataimport/incoming
```

Linux systems would use paths such as:

```
/tmp/dataimport/incoming
```

`execute.import` determines whether or not data imports execute in the environment. Its default is `false`.

`import.scheduler.startdelay` is a value, in milliseconds, that determines the interval from when the Quartz scheduler starts and the import process is executed for the first time. For example, a value of 60000 means that the scheduler is delayed for one minute.

`import.scheduler.repeatinterval` is a value, in milliseconds, that determines how often the import process is run.

`pricechange.scheduler.startdelay` and `pricechange.scheduler.repeatinterval` are used by Back Office only. They are the Quartz scheduler settings for the process that applies imported price changes when they are about to go into effect or expire.

`logger.filename` points to another properties file containing the string values that can be customized for DIMP messages.

dimplogger.properties

This is the file referred to by the value, `logger.filename`, in `spring.properties`. It contains text values that can be customized to make DIMP messages easily distinguishable in the `oracleretail` log file.

Every DIMP message appears with the `dimp.` prefix. `dimp.text1`, `dimp.text2` and `dimp.text3` are used depending on how much information is supplied by the underlying system.

Archive File Format

The Archive File is of the following format:

```
META-INF
  MANIFEST.MF
ItemImport-12345-20032-007.xml
PriceImport-12345-20032-007.xml
StoreHierarchy.xml
....
```

The suggested file naming convention for the archive is as follows:

```
[arbitrary_portion]-[store_id]-[YYYYMMDD]-[NNN].jar
```

Where `[arbitrary_portion]` can be used by the implementation team for any value, and `[NNN]` is the batch ID in the range of 0 through $2^{32}-1$, or 2,147,483,647 (because of the limitations of the XSD `int` datatype). This is a sequential number that is used to determine the processing order for archives, if more than one exists on the server at a time. The date is only available for visual reference. If the file name is not

formatted as above, the values in the manifest are used instead. However, if both the archive file name and the file names within the manifest contain a batch ID, the value in the archive file name takes precedence.

There is no restriction on the file names and they can be in any format. But the exact file names have to be listed in the MANIFEST.MF.

The format of the MANIFEST.MF is as follows

```
Manifest-Version: 1.0

# This manifest describes the contents of an archive referred to as a
# bundle. The following two values list the ID of the batch that
# produced this bundle and the ID of the destination store to receive
# it. The BatchID should be numeric less than 2^32-1.

BatchID: <N>
StoreID: <NNNNN>

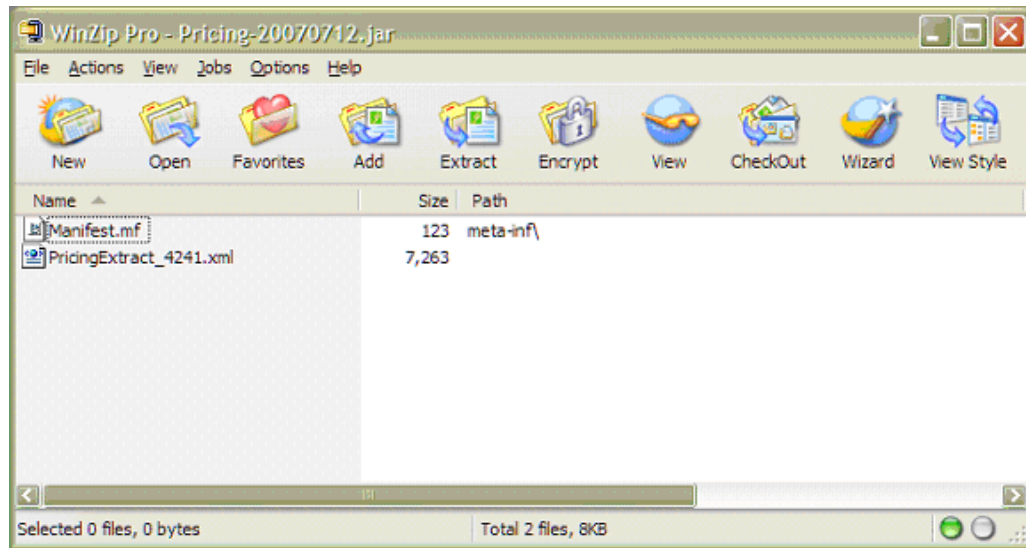
# The following section lists the files contained in this bundle archive.
# Each key should begin with "FileN" without quotes and N being a number.
# The value of the key consists of a bundle entry file name followed
# by hard brackets containing a list of files that should be processed
# before it.
#
# e.g. File1: ItemImport.xml[TaxImport.xml,StoreHierarchyImport.xml]
#
# The order of the files or their dependency list is not important.

File1: <filename1>[<optional dependencies>]
...
FileN: <filenameN>[<optional dependencies>]
```

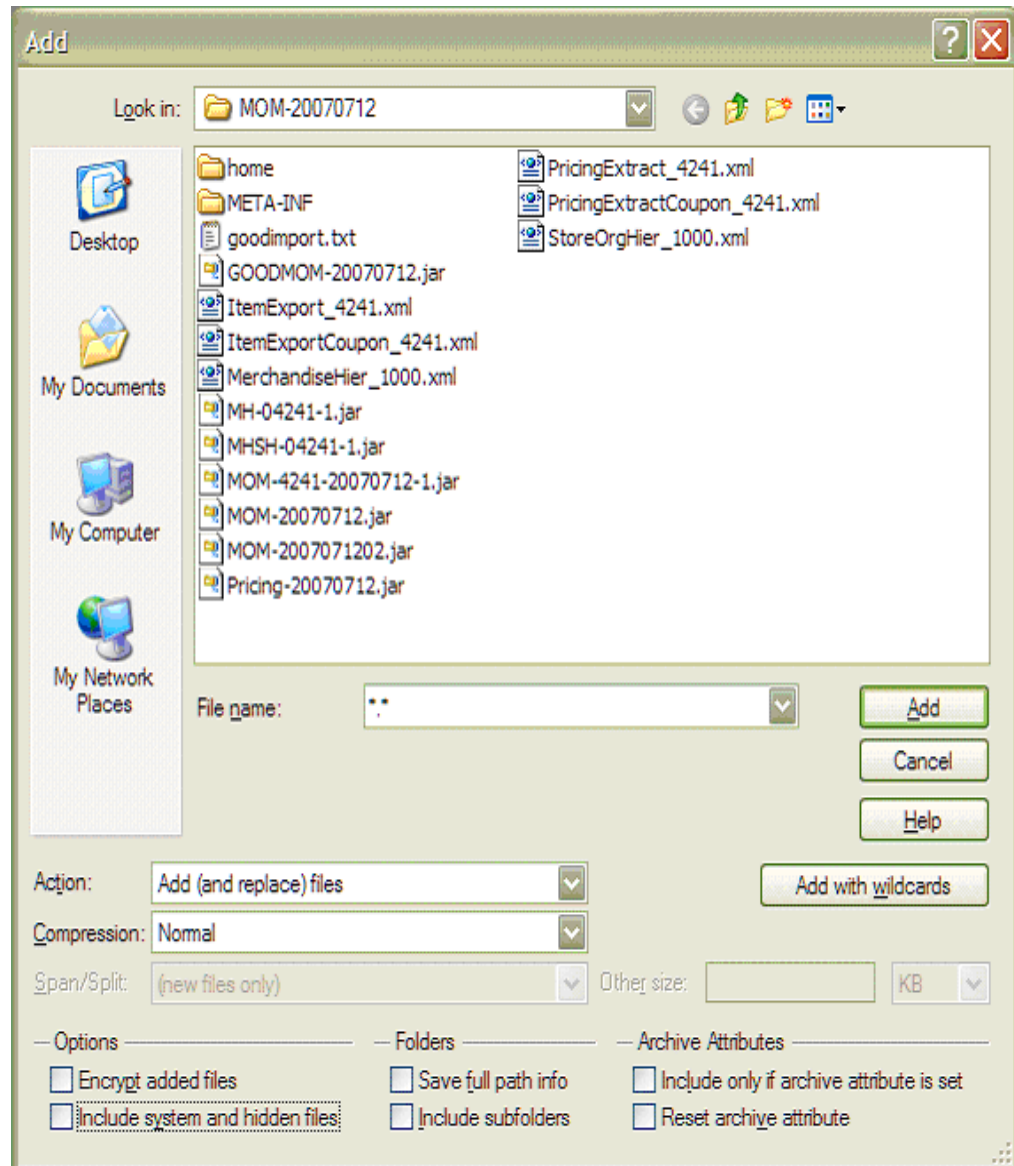
With the exception of manifest.mf, path names should not be used when creating the manifest. In the figure below, note that the path column is empty except for meta-inf, the path for manifest.mf.

Note: Although WinZip cannot be used to create a bundle, it can be used to inspect the bundle, as well as add, delete, or modify the XML contents. Use the following Jar command line utility to create a bundle:

```
C:\temp\dataimport\archive>%JAVA_HOME%\bin\jar -cvfm test_
coupon3.jar manifest_details.txt
PricingImportSample_addCouponDiscount.xml ItemImportSample_
addCoupon.xml
```

Figure 3–1 Adding Files To a Jar

In the following screen shot of the dialog box for adding files to a WinZip archive, note that the **Save full path info** option at the bottom is unchecked.

Figure 3–2 Adding Files To A WinZip Archive

Here is an example of a manifest file

```

Manifest-Version: 1.0

# This manifest describes the contents of an archive referred to as a
# bundle. The following two values list the ID of the batch that
# produced this bundle and the ID of the destination store to receive
# it. The BatchID should be numeric less than 2^32-1.

BatchID: 1
StoreID: 04241
File1: ItemImportSample_addCoupon.xml[]
File2: PricingImportSample_addCouponDiscount.xml[ItemImportSample_addCoupon.xml]

# The following section lists the files contained in this bundle archive.
# Each key should begin with "FileN" without quotes and N being a number.
# The value of the key consists of a bundle entry file name followed
# by hard brackets containing a list of files that should be processed
# before it.
#
# e.g. File1: ItemImport.xml[TaxImport.xml,StoreHierarchyImport.xml]
#
# The order of the files or their dependency list is not important.

File1: TaxImport.xml[]
File2: MerchandiseHierarchyImport.xml[]
File3:
ItemImport.xml[TaxImport.xml,MerchandiseHierarchyImport.xml,StoreHierarchyImport.x
ml]
File4: ItemImport2.xml[ItemImport.xml]
File5: PriceImport.xml[ItemImport2.xml]
File6: StoreHierarchyImport.xml[]
File7: EmployeeImport.xml[StoreHierarchyImport.xml]

```

Oracle Retail Merchandising System Configuration

If the retailer is integrating with Oracle Retail Merchandising System, it is assumed that the retailer is setting up items within Oracle Retail Merchandising System, and not using this feature in Back Office. If the retailer chooses to add or edit an item within Back Office, then that item data might be overridden by the next download from Oracle Retail Merchandising System.

Some data fields are defaulted to the values shown in [Table 3–3](#).

Table 3–3 Oracle Retail Merchandising System Default Values in the Back Office Item Maintenance Screen

Back Office Data Field	Default Value when integrating with Oracle Retail Merchandising System or Limitation
Cost	0
Class	Items belong to one class only
Manufacturer	Null
Planogram	Null
Labels/Tags Template Type	Default
Serialized	FALSE
Restocking Fee	FALSE
Activation Required	No
Registry Eligible	No

Table 3–3 Oracle Retail Merchandising System Default Values in the Back Office Item Maintenance Screen

Back Office Data Field	Default Value when integrating with Oracle Retail Merchandising System or Limitation
Special Order Eligible	No
Employee Discount Eligible	Yes
Damage Discount Eligible	Yes
Size Entry Required	No
Authorized for Sale	Active
Item Department	The first department in the drop down list. If no Item Department is specified, then the value is defaulted to the first value in the drop down list.

Service items (non-merchandise items that are non-inventory) need to be loaded separate from the download process.

In Oracle Retail Merchandising System, differentiators 1 through 4 are sent as values and are mapped to STYLE, COLOR, SIZE and NULL in Point-of-Service.

Oracle Retail Price Management Configuration

If the retailer is integrating with Oracle Retail Price Management, it is assumed that the retailer is setting up items within Oracle Retail Price Management, and not using this feature in Back Office. If the retailer chooses to add or edit an item within Back Office, that item data might be overridden by the next download from Oracle Retail Price Management.

Note: You must edit the Data Definition Language (DDL) before building the store's database when integrating with Oracle Retail Price Management.

In the files `CreateTableTemporaryPriceChangeItem.sql` and `CreateTablePriceDerivationRule.sql` there are the following two lines:

```
-- Uncomment and use this index for Oracle Retail Price Management
(RPM) integrations
-- CREATE UNIQUE INDEX ITM_TMP_PRM_IDX ON MA_ITM_TMP_PRC_CHN (ID_
PRM, ID_PRM_CMP, ID_PRM_CMP_DTL);
```

Remove the dashes that start the second line so that when the database is built, these three columns (that contain Oracle Retail Price Management IDs) create a unique index.

In the first phase of the integration, some data fields are defaulted to the values shown in [Table 3–4](#).

Table 3–4 Oracle Retail Price Management Default Values

Back Office Screen	Back Office Data Field	Default Value when integrating with Oracle Retail Price Management or Limitation
Discount Rule	Start Time	0:00
Discount Rule	End Time	23:59:59
Discount Rule	Source	Promotions defined at Item Level only
Discount Rule	Target	Promotions defined at Item Level only
Discount Rule	Source Threshold	None
Discount Rule	Source Limit	None
Discount Rule	Target Threshold	None
Discount Rule	Target Limit	None
Discount Rule	Number of Times Per Transaction	-1
Discount Rule	Accounting Method	Discount
Discount Rule	Allow Source to Repeat	Yes
Discount Rule	Deal Distribution	Target
Discount Rule	Target Quantity	1
Price Maintenance	Start Time	0:00
Price Maintenance	End Time	23:59:59
Price Maintenance	Status	Back Office gets status from effective date and record descriptors
Price Maintenance	Template Type	Default

Service items (non-merchandise items that are non-inventory) need to be loaded separate from the download process.

Data Requirements – Oracle Retail Strategic Store Solutions to Oracle Retail Sales Audit

For Point-of-Service there are two types of data that must be provided for the Point-of-Service to start, and before the task of ringing up items begins:

- Base Data – the basic configuration data. This includes:

- currency
- password policy
- user role
- work group information

This information is provided with the application.

- Store and Application Data (Seed Data) – this is data that differentiates one store from another. This information includes:

- item
- employee

- pricing rules
- tax rule

Examples of Base Data insert statements can be found in the build at `<root>\modules\utility\deploy\360common\db\sql\Base` and in the Point-of-Service installation at `\OracleRetailStore\Server\360common\db\sql\Base`.

Examples of Seed Data insert statements can be found in the build at `<root>\modules\utility\deploy\360common\db\sql\Seed` and in the Point-of-Service installation at `\OracleRetailStore\Server\360common\db\sql\Seed`.

Loading base data and Merchandise Operations Management bundles into the Store level database does not provide enough data for the Point-of-Service to initiate. This combination is missing some required data, such as the store's tax geocode and employees to log into the application. In order to provide a minimum amount of information, combine a small number of files from the Seed Data with all the files from Base Data. The additional files are:

- `InsertTableCodeList.sql` – data used in Point-of-Service drop down lists for pay-in/out, Tax overrides and so forth.
- `InsertTableEmployee.sql` – employee information.
- `InsertTableEmployeeHierarchyAssociation.sql` – employee information.
- `InsertTableStoreSafeTender.sql` – tenders used in the store.
- `InsertTableZTaxTables.sql` – base tax information, store geocode.
- `InsertTablePriceDerivationRuleType.sql` – price derivation rule types.

Capacity Planning

This section lists the approximate hard drive sizes that are required at each store to be able to support the Data Import project.

The following assumptions were made to arrive at an approximate capacity:

- The archival period is one week.
- The frequency is one file per day.
- The TAX Import file is not part of the Import Bundle.
- Peak Load for the EMPLOYEE Import is 30 employees per file.
- The Peak Load Capacity of each file is taken into consideration for the estimation.
- The average compression ratio in creating a jar file is considered to be 60%.
- As the frequency is one bundle per day, and the archival period is one week, therefore the maximum number of files on the disk is eight.
- A footprint on the DDI (Data Distribution Interface) on the Store Server is considered to be the size of one bundle and added to the final estimate. The footprint on the DDI is not part of the scope of the DIMP.
- Because the peak load size for Merchandise Hierarchy is not defined, a load of 5000 records is estimated.

Table 4–1 identifies the file sizes for components in the data import at a store.

Table 4–1 File Sizes

Type of Import	One-Record Size in Bytes	Peak Load (Number of Records)	Peak File Size in Bytes
Item	950.00	15,000,000.00	14,250,000,000.00
Pricing	1,600.00	820,000.00	1,312,000,000.00
Store	710.00	5,000.00	3,550,000.00
Merchandise	300.00	5,000.00	1,500,000.00
Employee	1,400.00	30.00	42,000.00

Total Size of Files

15,567,092,000.00 Bytes

Table 4–2 identifies the sizes of data import bundles.

Table 4–2 Bundle Size

Bundle Size (jar Size)	Assuming 60% Compression Ratio in creating a jar	9,340,255,200.00	Bytes
		8,900.00	MB
Approximate Bundle Size		8.69	GB

Table 4–3 identifies the required hard-drive capacities to enable a data import.

Table 4–3 Hard Drive Capacity

Seven files in Archive + One File in current	71,200.00	MB
	69.53	GB
Approximate Hard Drive Size to retain the Bundles	70.00	GB
Footprint on DDI Store Server (the DDI remains the responsibility of the implementation team to implement) - assuming size of one Bundle	78.69	GB

Required Hard Drive Capacity (Approximate)

80.00 GB

Table 4–4 Item Import Data Volumes

Data Volumes		
Item	800,000 – 1.5 million for peak season 5000 – 15,000 for delta	1.5 million
Item Location	See Item	See Item
Item (Merchandise) Hierarchy	number of departments	number of departments
	groups	number of groups
	number of hierarchies	number of hierarchies
Organizational (Store) Hierarchy	5000 stores, 6 levels	5000 stores, 6 levels
	number of regions	
	number of districts per region	
	number of stores per district.	
Tax data	See Item (since any tax info is limited to item related attributes such as tax group ID)	See Item (since any tax info is limited to item related attributes such as tax group ID)
	*Tax info does not come from Oracle Retail Merchandising System	*Tax info does not come from Oracle Retail Merchandising System

Pricing Import Data Volumes

Data Volumes: 800000 price changes per day per store.

Customization Notes

Data Import Extension Points and Development

Oracle Store Solutions has provided not only extension points for enhancing or modifying the capabilities of the existing data imports, but there are also tools provided for jump-starting an altogether new data import. Do the following to create a new data import module:

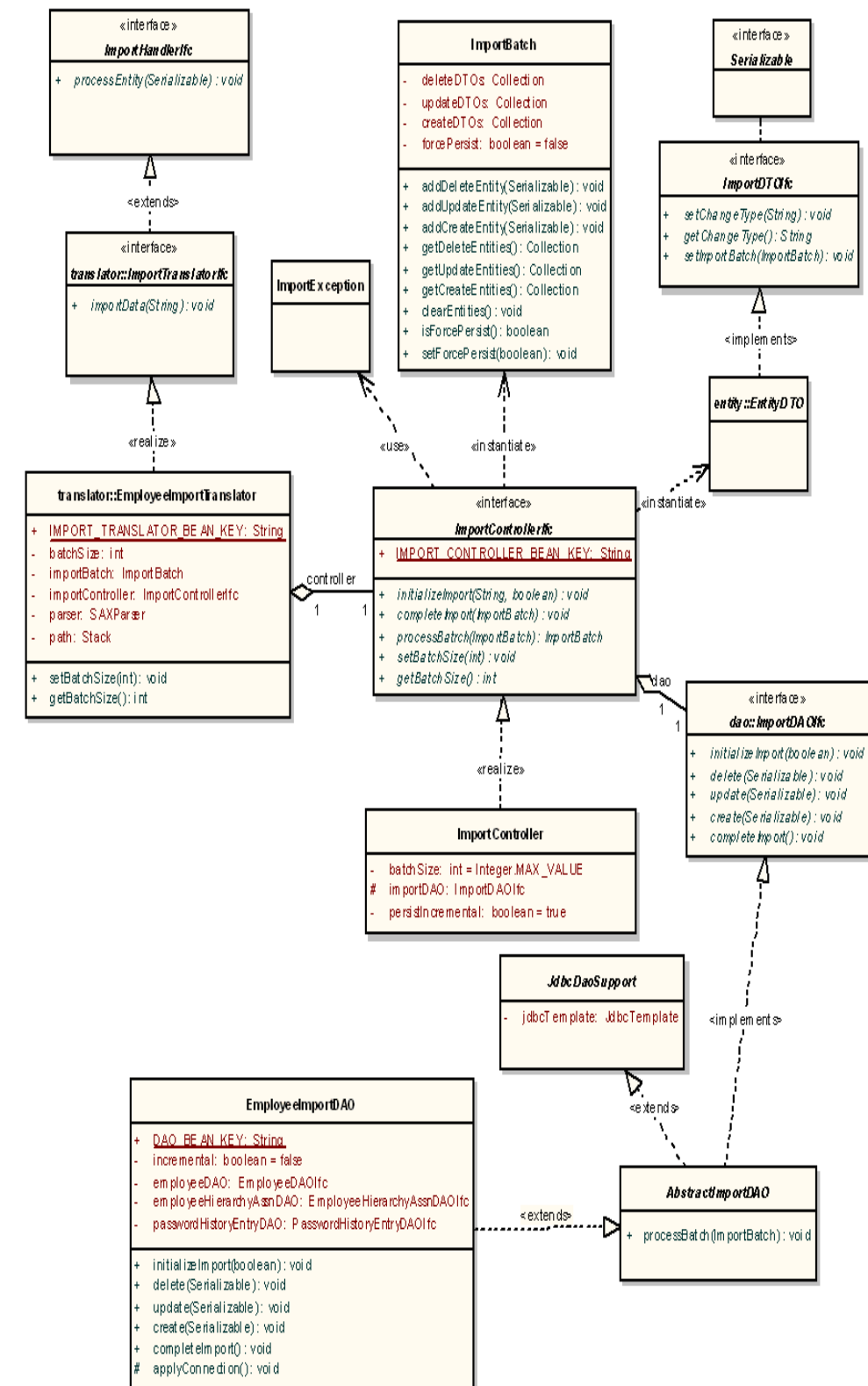
1. Compose XSD that the import data conforms to.
2. Generate sample XML based on the XSD. This can be done using the Eclipse EMF plug-in. See <http://www.eclipse.org/>
3. Map the XSD to the Data Model.
4. Create a Message Driven Bean and update the appropriate deployment descriptors.
5. Use SAXParserGenerator with XSD.
6. Use DAOGenerator to generate data access objects (DAO) for tables mapped to.
7. Rename DAO classes to match logical names of tables.
8. Delete duplicate DTOs or DAOs that might exist in other packages and that can be reused.
9. Update DAOIfc method parameters to pass actual DTO objects.
10. Remove column names from UPDATE_SQL that are not updated during update procedure from DAO and SQLIfc.
11. Update DAO get*Statement() methods to map DTO fields to PreparedStatement buckets.
12. Create a test that reads the XML and sends it to translator. How the XML is created or read is not important at this time, nor is using Spring or JUnit or AppServer.

The following sections discuss these steps in more detail. Where these steps overlap with steps for enhancement (as opposed to steps for creating new imports), the enhancement steps are identified.

First, extension points are identified, and techniques for enhancing existing data imports are described. Each of the previously mentioned DIMP modules (Taxation, Merchandise Hierarchy, Store Hierarchy, and Employee) follow the same patterns of implementation and vary in minor details only. We concentrate on Employee.

The following diagram is the Employee Data Import Static Model.

```
class Import Components
```



Import Adapter and Translator

The entry point for data imports is the `ImportIOAdapterIfc`. It is configured through a Spring context as either `EEImportIOAdapter`, for JCA implementations, or `FileImportIOAdapter` for direct file I/O implementations. The IO Adapter retrieves the bundles from the file system, determines the processing order, and passes the XML stream data to the `ImportInitiator`, which determines the import type from the payload and passes the string to a translator. The `ImportInitiator` (as the `BeanLocator`) provides an `ImportTranslatorIfc` from the service context by passing the key `EmployeeImportTranslator.IMPORT_TRANSLATOR_BEAN_KEY`, for example.

The following example shows the `EEImportIOAdapter` implementation in use:

```
<!-- Import IO Adapter Implements com._
360commerce.commerceservices.importdata.ImportIOAdapterIfc -->
<bean id="service_ImportIOAdapter" class="com._
360commerce.commerceservices.importdata.EEImportIOAdapter">
</bean>
<!--<bean id="service_ImportIOAdapter" class="com._
360commerce.commerceservices.importdata.FileImportIOAdapter">
```

SAXParserGenerator

If creating a new data import module and starting with a defined XSD, a simple utility can be run to generate code for a Translator, SAX handlers, simple DTO, and a skeleton Import DAO. The following is an example of how to run this utility.

Example 5-1 *SAXParserGenerator utility command prompt*

```
<root>\modules\utility>java
com._360commerce.codegen.importtranslator.SAXParserGenerator "C:\Data
Import\Design\Employee\EmployeeImport.xsd"
com._360commerce.commerceservices.employee.importdata
../../commerceservices/employee/src
```

This command line example shows that the utility program is Java-based and takes three arguments:

- The location of the XSD file.
- The desired package name for the generated source code.
- The directory in which to place new source code files.

This utility can be configured as an executable target in your favorite Integrated Development Environment (IDE) so this utility can be run again as changes continue to be made to the XSD which defines the format of the new data input.

The code generation uses the Java-based Velocity templates and APIs. See <http://jakarta.apache.org/velocity>. The templates can be found at `<root>/modules/utility/templates/`.

Manually Editing Generated Code

The generated code requires additional manual editing before it can be used. For example, the `ImportDAO` has only the barest of implementations in its methods. Add code to pass various DTOs to the correct DAO that can handle it.

Appropriate DTOs might already exist in the codebase. Examine the attributes of the pre-existing DTO to see if it or the generated DTO should be used. In some cases, additional code might need to be added. For example, if you consider that a single-entity DTO usually represents a single record in the database, the SAX handlers

are coded to not process child DTOs passed to the SAX handlers until the DTO that a SAX Handler creates is successfully processed.

Example 5-2 *EmployeeAccessHandler Process DTO Before Children*

```
/**
 * End handling this element. Calls {@link
 * ImportHandlerIfc#processEntity(java.io.Serializable)}
 * @throws SAXException
 */
public void end() throws SAXException
{
    try
    {
        // process this first
        parent.processEntity(employeeAccessDTO);

        // process all its children
        Iterator iter = children.iterator();
        while (iter.hasNext())
        {
            Serializable child = (Serializable)iter.next();
            parent.processEntity(child);
        }
    }
    catch (ImportException e)
    {
        logger.error("Could not end element " + getText(), e);
        throw new SAXException("Could not end element " + getText(), e);
    }
}
```

However, in some cases, such as when there are important attributes that are needed to fill the DTOs, and which need to be persisted immediately, the call to `parent.processEntity(Serializable)` can be commented out of the `end()` method and added to the `start(Attributes)` method. The `start(Attributes)` method is called when parsing the beginning of the XML element. Notice in the following example, the value for "Incremental" defaults to true if it does not exist.

Example 5-3 *EmployeeImportHandler Process DTO During Start*

```
/**
 * Start handling this element by inspecting its attributes, if any.
 * @param attributes the attributes given.
 * @throws SAXException
 */
public void start(Attributes attributes) throws SAXException
{
    String incremental = attributes.getValue("Incremental");
    Boolean bIncremental = (incremental != null)? Boolean.valueOf(incremental)
: Boolean.TRUE;

    employeeImportDTO.setEmployeeImportIncrementalAttribute(bIncremental.booleanValue(
));

    try
    {
        // process this first
        parent.processEntity(employeeImportDTO);
    }
}
```

```

        catch (ImportException e)
        {
            logger.error("Error starting import" + employeeImportDTO, e);
            throw new SAXException("Error starting import" + employeeImportDTO,
e);
        }
    }
}

```

There also might be a scenario where parent XML element values, such as IDs, are required for child DTO objects. These attributes might have to be added manually to the DTOs and set by the handlers. See the Merchandise Import DTO, LevelDTO as an example, and the handlers that call its set methods.

If it seems that the SAX handlers or the DTOs are missing attributes for defined XML elements, there might be errors in the XSD that the SAXParserGenerator cannot decipher. Ensure that your XSD validates properly based upon the schema at <http://www.w3.org/2001/XMLSchema>.

Metadata

The top-level element of each import includes metadata pertaining to the import bundle. Among other possible uses, this data is included in import bundle tracking and error logging. The following is an example XML fragment. Consult the development team for the status of data import schemas beyond this release.

```

<ItemImport
    Priority="0"
    FillType="FullIncremental"
    Version="1.0"
    Batch="1"
    CreationDate="2001-12-17T09:30:47.0Z"
    ExpirationDate="2007-12-17T09:30:47.0Z"
    xsi:noNamespaceSchemaLocation="ItemImport.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    . . .

```

The metadata attributes are defined as follows:

Priority

An integer specifying the order, from lowest to highest, in which multiple files of one type in a bundle should be processed.

FillType

The feed method: Kill And Fill, Delta Incremental, or Full Incremental. The XSD specifies which of these are allowed for an import type. For example, Tax allows only Kill And Fill, while Item allows all three.

Version

The version of the application processing the data.

Batch

An integer sequence number, corresponding to the ID of the process that created the file.

CreationDate

A timestamp identifying the file's creation time.

ExpirationDate

A timestamp beyond which a file has become stale and should not be processed.

ImportControllerIfc

The current implementation of the `ImportControllerIfc` operates well in most circumstances. However, there might be circumstances that call for a different version of the controller to be plugged in. For example, a new controller might put a parsed batch onto one of many secondary queues instead of passing it synchronously to a DAO, then returning control to the translator to continue parsing the import.

The secondary queue is another thread that takes the incoming batch and passes it to an instance of the import DAO. This enables multiple batches to be processed at once.

Strategic Store Solutions to Oracle Retail Sales Audit Extension Points and Development

There are three distinct situations in which a implementation team would need to extend the functionality in the Export File Generator:

- Adding data elements to the RTLog Format.
- Creating an entirely new fixed length export format.
- Creating an entirely new export format which is not fixed length.

Adding Data Elements to the RTLog Format

To add VAT information added to the one or more of the reference fields in the Transaction Item record to the RTLog a implementation team takes the following steps:

1. Define the format of the VAT data.
2. Depending on the outcome of step 1, it might be advantageous to modify the definition of a Reference field in the Transaction Item record. This cause the creation of Acme-specific Export Format Configuration file. If this is desirable, copy `RTLogFormat.xml` to `AcmeRTLogFormat.xml` and make the modifications in this file.
3. Define how the columns in the table `TR_LTM_SLS_RTN_TX` map to the format defined in step 1.
4. Write a `FieldMapper` class called `AcmeItemVATTax.java` to perform the mapping.
5. Copy `RTLogMappingConfig.xml` to `AcmeRTLogMappingConfig.xml` and make the following change to the new file:

```
<TABLE table="TR_LTM_SLS_RTN_TX">
    <MAP column="MO_TX_RTN_SLS" record="TransactionTax" field="TaxAmount"

    fieldMapper="com.acme.exportfile.RTLog.fieldmappers.AcmeItemVATTax" />
</TABLE>
```

6. Modify `StoreServerConduit.xml` to use `AcmeRTLogMappingConfig.xml` and `AcmeRTLogFormat.xml` instead of `RTLogMappingConfig.xml` and `RTLogFormat.xml`.

If the Reference field is partitioned correctly, and the values coming from the database to these new fields do not requires manipulation, then it is possible that the `FieldMapper` class is not required.

Creating a New Fixed Length Export Record Format

Currently, Oracle Retail has only one way to send transactional data to a customer's back end systems: POSLog. However, it is expensive and time consuming to extend POSLog, to explain it to customers and to develop the code that loads it into the customer back end.

It might be faster and cheaper to use the Export File Generator to generate the transaction log format that the customer is already consuming.

The generation of all three current formats (DTM for Central Office, POSLog for the customer backend, and RTLog for Oracle Retail Sales Audit) simultaneously has been tested in the development environment.

Here are the steps to create transaction log export code for "Acme", a generic customer:

1. Work with Acme developers to create a mapping document that describes the relationship between the Oracle database and the current Acme back end system/transaction log format. A mapping exercise of this type must be done even if the customer eventually chooses to use the POSLog to transfer the data. Understanding the customer's current transaction log can provide valuable insight into the data requirements.
2. Construct an Acme-specific Export Format Configuration file which describes all the records in the Acme transaction log; call this file AcmeTLogFormatConfig.xml.
3. Create an Acme-specific Mapping configuration file; call this file AcmeTLogMappingConfig.xml.
4. Create an Acme-specific Entity Reader configuration file; call this file AcmeTLogExtractConfig.xml.
5. If Acme exports the RTLog for Oracle Retail Sales Audit, the RTLogExportDaemonTechnician and RTLogExportDaemonThread can still be used to export the Acme Tlog formatted data. Just create another entry in StoreServerConduit.xml with a different technician and daemon name. This entry looks like the following:

```
<TECHNICIAN name="AcmeTLogExportDaemonTechnician"
            class="RTLogExportDaemonTechnician"
            package="com.extendyourstore.domain.manager.RTLog"
            export="Y">
  <PROPERTY propname="daemonClassName"
    propvalue="com.extendyourstore.domain.manager.RTLog.RTLogExportDaemonThread" />
  <PROPERTY propname="daemonName"
    propvalue="AcmeTLogExportDaemon" />
  .
  .
  .
</TECHNICIAN>
```

6. Modify StoreServerConduit.xml to use AcmeTLogExtractConfig.xml, AcmeTLogFormatConfig.xml and AcmeTLogMappingConfig.xml when exporting the Acme TLog.
7. Determine the batch ID column to use for this process. By convention, DTM uses TR_TRN.ID_TLOG_BTCH, POSLog uses TR_TRN.ID_BTCH_ARCH, and RTLog uses ID_RTLOG_BTCH. If your system exports RTLog, you must override RTLogExportBatchGenerator.retrieveTransactionList() and RTLogDatabaseAdapter.postResults() to change the column your application uses.

8. Over the course of development add table names to AcmeTLogExtractConfig.xml, mapping information to AcmeTLogMappingConfig.xml. Write Acme-specific FieldMapperIfc and AccessorIfc classes.
9. It is necessary to create an Acme-specific implementation for the MappingResultIfc interface to hold the Acme transactional information. Call this class AcmeTLogMappingResult. This necessitates the creation of an Acme-specific EntityMappingObjectFactoryIfc class. Call this class AcmeEntityMappingObjectFactory.
10. It is necessary to create an Acme-specific implementation for the RecordFormatContentBuilderIfc to assemble the Acme-specific export records. Call this class AcmeTLogRecordFormatContentBuilder. This necessitates the creation of an Acme specific RecordFormatObjectFactoryIfc class called AcmeRecordFormatObjectFactory.
11. Modify StoreServerConduit.xml to use the AcmeEntityMappingObjectFactory and the AcmeRecordFormatObjectFactory when exporting the Acme TLog.

Exporting a Non-Fixed-Length Record Format

There are other styles of text besides fixed record length which have been used to transfer transactional information to the enterprise. For example: comma delimited, and tag and value. To support either of these you must complete all the steps in the previous section, as well as the following:

1. It is likely that you need additional information about the export file format. As a result you must add information to the Export Format Configuration file, and create an Acme-specific implementation of the RecordFormatConfiguratorIfc interface; call this class AcmeRecordFormatConfigurator.
2. The FieldFormat class formats its data based on the data type and generates a fixed length field. When all the fields in a record are aggregated, this creates a fixed length record. This class must be replaced by an Acme-specific implementation; call this class AcmeCommaDelimitedFieldFormat. It might also be necessary to create an Acme-specific implementation of RecordFormatIfc; call this class AcmeCommaDelimitedRecordFormat.
3. Modify AcmeRecordFormatObjectFactory to return AcmeRecordFormatConfigurator, AcmeCommaDelimitedFieldFormat, and AcmeCommaDelimitedRecordFormat.

Object Factories

Object factories provide system implementers with the means to replace base product implementations with classes that are more appropriate to their needs. The object factory classes appear as entries in configuration files, and often times a configuration file functions as an object factory. This section discusses the object factory aspects and the configuration aspects of the configuration files.

StoreServerConduit.xml

The Store Server Conduit file (<root>\applications\pos\config\conduit\StoreServerConduit.xml) defines at runtime the classes and configuration files that make up the managers and technicians in the Point-of-Service Store Server. One of the technicians it defines is the RTLogExportDaemonTechnician. Following are the classes the Store Server Conduit file defines for use when exporting the RTLog:

Table 5–1 Store Server Conduit File

Class Name	Interface Name	Description
RTLogExportDaemonTechnician (com.extendyourstore.domain.manager.rtlog)	RTLogExportDaemonTechnicianIfc (com.extendyourstore.domain.manager.rtlog)	Sets up the RTLog Export Process. The Dispatcher instantiates this class and then sets all the other parameters this object. It is also responsible for managing the batch regeneration process.
RTLogExportDaemonThread (com.extendyourstore.domain.manager.rtlog)	RTLogExportDaemonThreadIfc (com.extendyourstore.domain.manager.rtlog)	Sleeps for a configurable amount of time, then wakes up and initiates the export process.
RTLogDatabaseAdapter (com.extendyourstore.domain.manager.rtlog)	DatabaseEntityAdapterIfc (oracle.retail.stores.exportfile)	Provides access to the database for reading each transaction Entity. This particular implementation uses the DataManager/ DataTechnician to retrieve this information.
RTLogEncryptingOutputAdapter (oracle.retail.stores.exportfile.rtlog)	OutputAdapterIfc (oracle.retail.stores.exportfile)	Writes the RTLog file to the configured directory. This particular adapter encrypts the file as it writes the file to disk. There is another adapter, RTLogOutputAdapter, which writes the file in clear text.
RTLogEncryptionAdapter (com.extendyourstore.domain.manager.rtlog)	EncryptionAdapterIfc (oracle.retail.stores.exportfile)	Provides access to the mechanisms for decrypting values which are encrypted in the database.
ExportFileConfiguration (oracle.retail.stores.exportfile)	ExportFileConfigurationIfc (oracle.retail.stores.exportfile)	Contains much the of configuration information in the RTLogExportDaemonTechnician; the technician passes this object to the daemon, which passes it to the batch generator which passes it to the export file generator.
RTLogExportFileResultAuditLog (com.extendyourstore.domain.manager.rtlog)	ExportFileResultAuditLogIfc (oracle.retail.stores.exportfile)	Formats the export result information for logging.
EntityMappingObjectFactory (oracle.retail.stores.exportfile)	EntityMappingObjectFactoryIfc (oracle.retail.stores.exportfile)	Instantiates the classes used to map the database Entity to the export file format.
RecordFormatObjectFactory (oracle.retail.stores.exportfile)	RecordFormatObjectFactoryIfc (oracle.retail.stores.exportfile)	Instantiates the classes used to setup and generate the export the file format.
ExtractorObjectFactory (com.oracle.xmlreplication)	ExtractorObjectFactoryIfc (com.oracle.xmlreplication)	Instantiates the classes used to generate the database Entity.
RTLogCurrencyAdapter (com.extendyourstore.domain.manager.rtlog)	CurrencyAdapterIfc (oracle.retail.stores.exportfile)	Provides currency services.

DomainObjectFactory

The DomainObjectFactory instantiates the RTLogExportBatchGeneratorIfc class. The RTLogExportBatchGenerator builds the WorkUnit (the list of transactions to export) and calls the WorkUnitController (ExportFileGenerator).

RTLogExportBatchGenerator also instantiates the ExportFileGeneratorIfc and the WorkUnitIfc. If you need a different implementation of either class, create a new implementation of RTLogExportBatchGenerator.

ExtractorObjectFactory

The ExtractorObjectFactory instantiates the classes that generate the database Entity class.

One item of note is that the application gains access to this factory through a singleton called ReplicationObjectFactoryContainer. All changes made to these classes must work for both DTM and Export File generation.

EntityMappingObjectFactory

The following table is a list of the classes this factory instantiates:

Table 5–2 EntityMappingObjectFactory Classes

Class Name	Interface Name	Description
MappingCatalogConfigurator (oracle.retail.stores.exportfile.mapper)	MappingCatalogConfiguratorIfc (oracle.retail.stores.exportfile.mapper)	Reads the mapping configuration file and builds an EntityMappingCatalogIfc object.
EntityMappingCatalog (oracle.retail.stores.exportfile.mapper)	EntityMappingCatalogIfc (oracle.retail.stores.exportfile.mapper)	Holds the information that describes the relationship between the tables and columns in the database to the records and fields in the export file. It contains a list of TableMaps and a map of Accessors.
TableMap (oracle.retail.stores.exportfile.mapper)	TableMapIfc (oracle.retail.stores.exportfile.mapper)	Contains a list of ColumnMaps associated with a table.
ColumnMap (oracle.retail.stores.exportfile.mapper)	ColumnMapIfc (oracle.retail.stores.exportfile.mapper)	Describes the relationship between a column and a field in a specific export record. It can contain a ValueMapping Hashmap and/or FieldMapper class to perform more complex mapping actions.
EntityMapper (oracle.retail.stores.exportfile.mapper)	EntityMapperIfc (oracle.retail.stores.exportfile.mapper)	Controls the mapping process. It stores the result in the MappingResultIfc object.
RTLogMappingResult (oracle.retail.stores.exportfile.rtlog)	MappingResultIfc (oracle.retail.stores.exportfile.mapper)	Contains the result of Mapping an Entity to the Export File Format.

RTLogMappingConfig.xml

This configuration file is a factory for FieldMapperIfc and AccessorIfc classes.

The simplest mapping occurs when a value goes directly from a column to a field. However, many times the mapping between a column and a field is more complex. If code is required, the configuration file calls out a FieldMapperIfc class to perform this mapping task. A FieldMapperIfc is associated with a particular table/column record/field mapping.

The values in a particular record are built up by processing of each individual ColumnMapIfc objects. There is no guarantee that all the data for a particular export record resides in a single row in the database. In fact it is unlikely. For example, a row from the Tender Line Item Table supplies the tender amount, but a row from the Credit Debit Tender Line Item Table supplies authorization information. Much processing can take place in between the time that the application has access to each of these rows.

An AccessorIfc object knows how to locate a particular existing “working” export record in the MappingResultIfc object. If a record is not available, the AccessorIfc creates a new one and store it in the MappingResultIfc object.

RecordFormatObjectFactory

Following is a list of the classes this factory instantiates:

Table 5–3 RecordFormatObjectFactory Classes

Class Name	Interface Name	Description
FieldFormat (oracle.retail.stores.exportfile.formater)	FieldFormatIfc (oracle.retail.stores.exportfile.formater)	Contains the attributes associated with a field including name, value, starting index, length, and data type.
RecordFormat (oracle.retail.stores.exportfile.formater)	RecordFormatIfc (oracle.retail.stores.exportfile.formater)	Contains a list of FieldFormatIfc objects.
RecordFormatCatalog (oracle.retail.stores.exportfile.formater)	RecordFormatCatalogIfc (oracle.retail.stores.exportfile.formater)	Contains a list of RecordFormatIfc objects.
RecordFormatConfigurator (oracle.retail.stores.exportfile.formater)	RecordFormatConfiguratorIfc (oracle.retail.stores.exportfile.formater)	Reads the format configuration file and builds a RecordFormatCatalogIfc object.
RTLogRecordFormatContentBuilder (oracle.retail.stores.exportfile.rtlog)	RecordFormatContentBuilderIfc (oracle.retail.stores.exportfile.formater)	Converts MappingResultsIfc object into the text that is written to the export file.
RTLogItemContainedRecords (oracle.retail.stores.exportfile.rtlog)	ContainedRecordsIfc (oracle.retail.stores.exportfile.formater)	A list of records, such as discounts, that are a part of the item information.
RTLogTransactionContainedRecords (oracle.retail.stores.exportfile.rtlog)	ContainedRecordsIfc (oracle.retail.stores.exportfile.formater)	A list of records, such as header total records, that are part of a transaction.

Configuration

Each of the configuration files used by this feature (Store Server Conduit, Entity Reader Configuration, Mapping Configuration, and Record Format Configuration) has already been referred to in this document. This section describes them in more detail.

The Store Server Conduit File

The Store Server Conduit file (<root>\applications\pos\config\conduit\StoreServerConduit.xml) defines the following settings for the RTLog Export process.

Table 5–4 Store Server Conduit File

Setting Name	Installed Product Value	Description
sleepInterval	600 (seconds)	The length of time between each execution of the RTLog export process.
exportDirectoryName	POSLog	The directory where the RTLog is placed.
formatConfigurationFileName	../config/rtlog/RTLogFormat.xml	The relative or absolute path of the Export Format configuration file.

Table 5–4 Store Server Conduit File

Setting Name	Installed Product Value	Description
entityReaderConfigurationFileName	../config/rtlog/RTLogExtra ctConfig.xml	The relative or absolute path of the Entity Reader configuration file.
entityMappingConfigurationFileName	../config/rtlog/RTLogMappi ngConfig.xml	The relative or absolute path of the Mapping configuration file.
maximumTransactionsToExport	-1	The maximum number of transactions that should exported to single RTLog file. The value -1 indicates there is not limit on the maximum number.

The Export Format Configuration file

The export format configuration file describes each of the export record types. For example, the RTLog specifies the following records:

- File Header
- File Tail
- Transaction Header
- Transaction Tail
- Transaction Item
- Item Discount
- Transaction Tax
- Transaction Tender

The following is a snippet from RTLogFormat.xml:

```
<?xml version="1.0"?>
<RECORD_FORMATS ... >
  <COMMENT>This file defines the format of the Oracle Retail Sales Audit
  RTLOG</COMMENT>
  <RECORD_FORMAT_VERSION version="V.12.0.5"/>
  <RECORD_FORMAT name="FileHeader">
    <FIELD_FORMAT name="FileRecordDescriptor" type="char" length="5"
value="FHEAD"/>
    <FIELD_FORMAT name="FileLineIdentifier" type="integer"
length="10"/>
    <FIELD_FORMAT name="FileType" type="char" length="4" value="RTLG"/>
    <FIELD_FORMAT name="FileCreateDate" type="datetime" length="14"/>
    <FIELD_FORMAT name="BusinessDate" type="date" length="8"/>
    <FIELD_FORMAT name="LocationNumber" type="char" length="10"/>
    <FIELD_FORMAT name="ReferenceNumber" type="char" length="30"
value=" " />
  </RECORD_FORMAT>
  .
  .
  .
</RECORD_FORMATS>
```

This snippet shows one Record definition (the File Header) composed of seven fields of various types, lengths and default values.

The Entity Reader Configuration File

This file defines tables that Entity Reader reads.

The Mapping Configuration File

This file describes the relationship between the tables and columns in the database and the records and fields in the export format. The following is a snippet from RTLogMappingConfig.xml:

```
<?xml version="1.0"?>
<ENTITY_MAPPER ... >
  <COMMENT>This is a configuration file for the Point-of-Service Transaction to
RTLog Mapping</COMMENT>
  <TABLE table="TR_TRN">
    <MAP column="DC_DY_BSN" record="FileHeader" field="BusinessDate"

fieldMapper="oracle.retail.stores.exportfile.rtlog.fieldmappers.BusinessDateMapper
"/>
    <MAP column="ID_STR_RT" record="FileHeader" field="LocationNumber"

fieldMapper="oracle.retail.stores.exportfile.rtlog.fieldmappers.StoreNumberMapper"
/>
    <MAP column="TS_TRN_END" record="TransactionHeader"
field="RegisterTransactionDate"

fieldMapper="oracle.retail.stores.exportfile.rtlog.fieldmappers.DateTimeMapper"/>
    .
    .
    .
    <MAP column="TY_TRN" record="TransactionHeader" field="TransactionType"
mappingStrategyOrder="FieldMapperThenValueMapping"

fieldMapper="oracle.retail.stores.exportfile.rtlog.fieldmappers.ExportItemsAndTaxS
tatusMapper">
      <VALUE_MAPPINGS handleNotFound="error">
        <VALUE_MAPPING DatabaseValue="1" RecordValue="SALE"/>
        <VALUE_MAPPING DatabaseValue="2" RecordValue="RETURN"/>
        <VALUE_MAPPING DatabaseValue="3" RecordValue="PVOID"/>
        .
        .
        .
      </VALUE_MAPPINGS>
    </MAP>
    .
    .
    .
  </TABLE>
  .
  .
  .
  <ACCESSOR record="FileHeader"

class="oracle.retail.stores.exportfile.rtlog.accessors.AccessFileHeader"/>
  <ACCESSOR record="TransactionHeader"

class="oracle.retail.stores.exportfile.rtlog.accessors.AccessTransactionHeader"/>
  .
  .
  .
</ENTITY_MAPPER>
```

Looking at this snippet, it is easy to see that the column TR_TRN.DC_DY_BSN maps to the `BusinessDate` field in the `FileHeader` record using the `BusinessDateMapper` class to format the data.

Also note that application uses a `VALUE_MAPPINGS` element to transform the value from the column TR_TRN.TY_TRN to equivalent value in the `TransactionType` field in the `TransactionHeader` record.

Development and Testing Tools

There are a number of tools that were developed during the course of this project that are helpful when extending this subsystem.

Classes

The following classes are all located at `<root>\modules\exportfile\src\oracle\retail\stores\exportfile\utility`:

Table 5–5 Exportfile Utility Classes

Class Name	Description
ExportTestDriver	<p>This class is a test harness that can be used to develop the configuration files, FieldMapperIfc and AccessorIfc classes in isolation from the rest of the application. It uses the classes DatabaseEntityAdapterTest, EncryptionAdapterTest, CurrencyAdapterTest, OutputAdapterTest and ExportFileResultAuditLogTest to emulate system specific adapters.</p> <p>An Eclipse-run configuration for this class should run out of the exportfile project. The classpath should the domain, foundation-client, foundation-server, common, utility, foundation-shared, clientinterfaces, datareplication projects and /thirdparty/apache-ant-1.6.2/lib/xml-apis.jar, /thirdparty/apache-ant-1.6.2/lib/xercesImpl.jar, and /thirdparty/apache/log4j-1.2.8.jar. It should also include the JDBC jar(s) for the database you are using.</p> <p>You most likely have to modify this class to use the appropriate JDBC driver, username, password and transaction IDs.</p>
FileDecryptionUtility	<p>By default the application (not the test harness) generates encrypted files. This class reads all the encrypted files from a target directory, decrypt them, and write them to a single target file. The main() method has three command line parameters:</p> <ul style="list-style-type: none"> ■ EncryptedDirectoryName - the pathname of the directory of *.ENC files ■ DecryptedFileName - the pathname of the decrypted file ■ EncryptionKey - the optional encryption key (uses default if not entered)
RTLOGReportDriver	<p>This class reads an export format configuration file and an export log file then generates a report file (rtlog_rpt.txt) to the current directory. This saves a lot effort when trying to determine if an export file has the correct data in it. The main() method has two command line parameters:</p> <ul style="list-style-type: none"> ■ ExportFileName - full/relative path pathname of the export file. ■ XMLFormatFileName - full/relative path pathname of the format file.

Executables in the bin Directory

The following BAT files are all located at <root>\modules\exportfile\bin:

Table 5–6 *bin Directory BAT Files*

Class Name	Description
setenv.bat	Sets up the classpath
RTLogFileDecryption.bat	Executes FileDecryptionUtility.class; it points at the bin\POSLog directory in the default installation, writes the decrypted records to RTLOG.DEC, and uses the default encryption key.
RTLogReport.bat	Executes RTLogReportDriver.class; it reads RTLOG.DEC, and uses to the export format file ..\config\RTLogFormat.xml.

Known Issues and Troubleshooting

DepartmentDefaultTaxGroup

When integrated with Oracle Retail Merchandising System, the `PreloadData > POSDepartment > DepartmentDefaultTaxGroup` field in the `MerchandiseHierarchyImport` is defaulted to 0 (zero). It is the responsibility of the implementation team to update this value in the bundle with a real `TaxGroup` ID for the items in question before the bundle reaches Strategic Store Solutions. Otherwise, a primary key violation might occur if zero is not an actual `TaxGroup` ID in the UDM.

Character Restrictions for UOMs

Retailers are restricted to only creating and using items with 2 character UOMs (Unit of Measure) as part of this integration.

Merchandise Operations Management transforms EA (Each) to UN (Unit) for the UOM in Item extracts to Strategic Store Solutions.

Strategic Store Solutions does not transform any other UOM in RTLogs to Merchandise Operations Management.

Oracle Retail Point-of-Service translates UN back to EA for the RTLog.

POSlog

For more information about the POSlog, see "POSlog Import Service" in the *Oracle Retail Central Office Operations Guide Release* and in the *Oracle Retail Back Office Operations Guide Release*.

Preload Section of ItemImport

Data in the Preload section of `ItemImport` is treated as an UPS which stands for **Upsert**. DIMP tries to Update data and if fails to update, then it Inserts data.

UTF-8

UTF-8 is a required character set for the database. DIMP supports multi-byte characters in the XML and puts this data into the database as UTF-8 character set.

Third-party Tax and Employee Information

Currently, all third-party Tax and Employee information must be presented in a specific file format for consumption by Central Office.

Implementation team need to be aware of this file format.

Tax and Employee files each have an XML Schema Definition just like other DIMP:

Example 6–1 Tax File XML Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="TaxImport" type="TaxImport">
    <xs:annotation>
      <xs:documentation>
        Copyright (c) 2006, Oracle. All Rights Reserved.
        XML Schema for data import of Tax Information. For Oracle Retail
        Store and Enterprise Applications.
        Contains Tax Authorities, Taxable Groups, Tax Rules and Rates
        data.
      </xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="TaxImport">
    <xs:sequence>
      <xs:element name="GEOCode" type="GEOCode" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="GEOTaxJurisdiction" type="GEOTaxJurisdiction" minOccurs="0"
        maxOccurs="unbounded"/>
      <xs:element name="TaxAuthority" type="TaxAuthority" minOccurs="0"
        maxOccurs="unbounded"/>
      <xs:element name="TaxableGroup" type="TaxableGroup" minOccurs="0"
        maxOccurs="unbounded"/>
      <xs:element name="TaxGroupRule" type="TaxGroupRule" minOccurs="0"
        maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="FillType" type="FillType" use="required" fixed="KillAndFill"/>
    <xs:attribute name="CreationDate" type="xs:dateTime"/>
    <xs:attribute name="ExpirationDate" type="xs:dateTime"/>
    <xs:attribute name="Version" type="xs:string"/>
    <xs:attribute name="Priority" type="xs:int"/>
    <xs:attribute name="Batch" type="xs:int"/>
  </xs:complexType>
  <xs:complexType name="TaxAuthority">
    <xs:sequence>
      <xs:element name="TaxAuthorityID" type="xs:integer"/>
      <xs:element name="TaxAuthorityName" type="xs:string"/>
      <xs:element name="RoundingCode">
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:minInclusive value="1"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="RoundingDigitsQuantity" type="xs:integer" minOccurs="0"/>
      <xs:element name="AddressLine" type="xs:string"/>
      <xs:element name="City" type="xs:string"/>
      <xs:element name="State" type="xs:string"/>
      <xs:element name="PostalCode" type="xs:string"/>
      <xs:element name="CountryCode" type="xs:string"/>
      <xs:element name="GeoCodeID" type="xs:string" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

```

</xs:sequence>
</xs:complexType>
<xs:complexType name="TaxableGroup">
<xs:sequence>
<xs:element name="TaxGroupID" type="xs:integer"/>
<xs:element name="TaxGroupName" type="xs:string" minOccurs="1" maxOccurs="1"/>
<xs:element name="TaxGroupDescription" type="xs:string"/>
<xs:element name="ReceiptPrintCode" type="xs:integer" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="TaxGroupRule">
<xs:sequence>
<xs:element name="TaxAuthorityID" type="xs:integer"/>
<xs:element name="TaxGroupID" type="xs:string"/>
<xs:element name="TaxTypeID" type="xs:integer"/>
<xs:element name="TaxTypeName" type="xs:string" minOccurs="0"/>
<xs:element name="TaxHolidayFlag" type="xs:boolean"/>
<xs:element name="TaxRuleName" type="xs:string"/>
<xs:element name="TaxRuleDescription" type="xs:string"/>
<xs:element name="CompoundRateSequenceNumber" type="xs:integer" minOccurs="0"/>
<xs:element name="TaxOnGrossAmountFlag" type="xs:boolean" minOccurs="0"/>
<xs:element name="CalculationMethodCode" minOccurs="0"/>
<xs:simpleType>
<xs:restriction base="xs:NMTOKEN">
<xs:enumeration value="LineItem"/>
<xs:enumeration value="Transaction"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="TaxRateRuleUsageCode">
<xs:simpleType>
<xs:restriction base="xs:NMTOKEN">
<xs:enumeration value="PercentageOrAmount"/>
<xs:enumeration value="DeriveFromTaxTable"/>
<xs:enumeration value="UseThresholdAmount"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="InclusiveTaxFlag" type="xs:boolean"/>
<xs:element name="TaxRateRule" type="TaxRateRule" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="TaxRateRule">
<xs:sequence>
<xs:element name="RateTypeCode" minOccurs="0">
<xs:simpleType>
<xs:restriction base="xs:NMTOKEN">
<xs:enumeration value="Percentage"/>
<xs:enumeration value="Amount"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:choice>
<xs:element name="TaxAmount" type="Currency"/>
<xs:element name="TaxPercentageRate">
<xs:simpleType>
<xs:restriction base="xs:decimal">
<xs:fractionDigits value="5"/>
</xs:restriction>
</xs:simpleType>

```

```
</xs:element>
</xs:choice>
<xs:element name="TaxAboveThresholdAmountFlag" minOccurs="0">
  <xs:simpleType>
    <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="TaxAboveThresholdAmount" />
      <xs:enumeration value="TaxEntireAmount" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="ThresholdAmount" type="Currency" minOccurs="0"/>
<xs:element name="TaxRateEffectiveTimestamp" type="xs:dateTime" minOccurs="0"/>
<xs:element name="TaxRateExpirationTimestamp" type="xs:dateTime" minOccurs="0"/>
<xs:element name="MinimumTaxableAmount" type="Currency" minOccurs="0"/>
<xs:element name="MaximumTaxableAmount" minOccurs="0">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:fractionDigits value="2" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="GEOCode">
  <xs:sequence>
    <xs:element name="GeoCodeID" type="xs:string"/>
    <xs:element name="TaxJurisdictionName" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="GEOTaxJurisdiction">
  <xs:sequence>
    <xs:element name="GeoCodeID" type="xs:string"/>
    <xs:element name="PostalCode" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="Currency">
  <xs:restriction base="xs:decimal">
    <xs:fractionDigits value="2" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="FillType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="KillAndFill"/>
  </xs:restriction>
</xs:simpleType>
</xs:schema>
```

Example 6–2 Employee File XML Schema Definition

```

<?xml version="1.0" encoding="windows-1252" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <!-- $Log:$ -->

  <xs:annotation><xs:documentation>
    Employee Import Schema. Copyright 2006 Oracle. All rights reserved.
  </xs:documentation></xs:annotation>

  <xs:element name="EmployeeImport" type="EmployeeImport">
    <xs:annotation><xs:documentation>
      Top-level element holding a collection of Employee elements.
    </xs:documentation></xs:annotation>
  </xs:element>

  <xs:complexType name="EmployeeImport">
    <xs:sequence>
      <xs:element name="Employee" type="EmployeeType" minOccurs="1"
maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="FillType" type="FillType" use="required"/>
  <xs:attribute name="CreationDate" type="xs:dateTime"/>
  <xs:attribute name="ExpirationDate" type="xs:dateTime"/>
  <xs:attribute name="Version" type="xs:string"/>
  <xs:attribute name="Priority" type="xs:int"/>
  <xs:attribute name="Batch" type="xs:int"/>
  </xs:complexType>

  <xs:complexType name="EmployeeType">
    <xs:annotation><xs:documentation>
      Represents a single employee's information.
    </xs:documentation></xs:annotation>
    <xs:sequence>
      <xs:element name="ChangeType" type="ChangeType" default="ADD"
minOccurs="1" maxOccurs="1" />
      <xs:element name="EmployeeID" type="ID" minOccurs="1" maxOccurs="1" />
      <xs:element name="EmployeeFirstName" type="xs:string" minOccurs="0"
maxOccurs="1" />
      <xs:element name="EmployeeLastName" type="xs:string" minOccurs="0"
maxOccurs="1" />
      <xs:element name="EmployeeMiddleName" type="xs:string" minOccurs="0"
maxOccurs="1" />
      <xs:element name="EmployeeFullName" type="xs:string" minOccurs="0"
maxOccurs="1" />
      <xs:element name="EmployeeSSN" type="SSN" minOccurs="0" maxOccurs="1"
/>
      <xs:element name="EmployeeRole" type="xs:string" minOccurs="0"
maxOccurs="1" />
      <xs:element name="PartyID" type="xs:int" minOccurs="0" maxOccurs="1"
/>
      <xs:element name="StatusCode" type="StatusCode" minOccurs="0"
maxOccurs="1" />
      <xs:element name="Locale" type="ID" minOccurs="0" maxOccurs="1" />
      <xs:element name="EmployeeAccess" type="EmployeeAccess" minOccurs="0"
maxOccurs="1" />
      <xs:element name="EmployeeType" type="StatusCode">
        <xs:annotation><xs:documentation>
          0 means 'Standard' employee, 1 means Temporary employee

```

```
        </xs:documentation></xs:annotation>
      </xs:element>
      <xs:element name="NumberDaysValid" type="xs:int" minOccurs="0"
maxOccurs="1">
        <xs:annotation><xs:documentation>
          Only applies to temporary employee
        </xs:documentation></xs:annotation>
      </xs:element>
      <xs:element name="TempEmployeeExpirationDate" type="xs:date"
minOccurs="0" maxOccurs="1">
        <xs:annotation><xs:documentation>
          Only applies to temporary employee
        </xs:documentation></xs:annotation>
      </xs:element>
      <xs:element name="EmployeeStoreOrHierarchyAssn"
type="EmployeeStoreOrHierarchyAssn" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="ID">
    <xs:restriction base="xs:string">
      <xs:maxLength value="10" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="SSN">
    <xs:restriction base="xs:string">
      <xs:maxLength value="9" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="StatusCode">
    <xs:restriction base="xs:string">
      <xs:enumeration value="0" />
      <xs:enumeration value="1" />
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="EmployeeAccess">
    <xs:annotation><xs:documentation>
      Holds all information regarding access to the system.
    </xs:documentation></xs:annotation>
    <xs:sequence>
      <xs:element name="EmployeeLoginID" type="xs:string" />
      <xs:element name="AccessPassword" type="xs:string" />
      <xs:element name="WorkGroupID" type="xs:int" />
      <xs:element name="EmployeeAltID" type="xs:string" minOccurs="0"
maxOccurs="1" />
      <xs:element name="NewPasswordRequired" type="xs:boolean" />
      <xs:element name="PasswordCreationDate" type="xs:dateTime" />
      <xs:element name="PasswordHistory" type="PasswordHistory"
minOccurs="0" maxOccurs="1">
        </xs:element>
      </xs:sequence>
    </xs:complexType>

  <xs:complexType name="PasswordHistory">
    <xs:sequence>
      <xs:element name="PasswordHistoryEntry" type="PasswordHistoryEntry"
minOccurs="1" maxOccurs="unbounded" />
```



```

    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="PasswordHistoryEntry">
    <xs:annotation><xs:documentation>
      Holds a single password history entry.
    </xs:documentation></xs:annotation>
    <xs:sequence>
      <xs:element name="PasswordCreationDate" type="xs:dateTime" />
      <xs:element name="AccessPassword" type="xs:string" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="EmployeeStoreOrHierarchyAssn">
    <xs:annotation><xs:documentation>
      Holds an employee association to a store and/or a hierarchy node.
      Generally, only one of the
      enclosed elements is provided; however, there may be cases where an
      employee needs both a store
      association and a hierarchy association, so a sequence with optional
      elements is used instead of
      a choice.
    </xs:documentation></xs:annotation>
    <xs:sequence>
      <xs:element name="EmployeeStoreID" type="RetailStoreId" minOccurs="0"
maxOccurs="1" />
      <xs:element name="EmployeeHierarchyAssn" type="EmployeeHierarchyAssn"
minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="EmployeeHierarchyAssn">
    <xs:sequence>
      <xs:element name="NodeID" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="NodeType" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="StoreGroupFunctionID" type="xs:int" minOccurs="1"
maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="ChangeType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="ADD" />
      <xs:enumeration value="UPD" />
      <xs:enumeration value="DEL" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="RetailStoreId">
    <xs:annotation><xs:documentation>
      Store Id's can only be five characters long and preferably only
      numerals.
    </xs:documentation></xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"></xs:minLength>
      <xs:maxLength value="5"></xs:maxLength>
    </xs:restriction>
  </xs:simpleType>

```

```
<xs:simpleType name="FillType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="KillAndFill"/>
    <xs:enumeration value="FullIncremental"/>
  </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Glossary

Batch

A collection of data operations that are processed at one time. The size is determined by a configurable parameter.

Bundle

A collection of import files, one file per data type, stored as a compressed file containing a manifest. It is expected that the retailer or implementation team is responsible for packaging and delivering to the Store the bundle along with manifest for all data feeds to the Store

Corporate

Used interchangeably with *enterprise*. The enterprise environment of the retailer where enterprise applications are deployed. Oracle Retail Central Office is deployed in the enterprise.

Data Access Object (DAO)

A class that can retrieve and persist data to and from a data source.

Data Distribution Infrastructure (DDI)

The infrastructure and application components that are responsible for distributing seed data from enterprise applications to Store applications, ODS at Corporate (or enterprise), and Store Database at the stores.

Data Transfer Object (DTO)

A class that contains data records from a received payload. The DTO's attributes are populated with the parsed data.

DIMP

Data Import

Incremental

There are two types of update operation, full incremental and delta incremental. Full incremental assumes that all the fields for a data type are supplied in the XML. A delta incremental import contains only the fields that are being changed.

ISP

In-Store-Processor

J2EE

Java 2 Enterprise Edition is a set of APIs designed to support tier 1 type business models.

Java Database Connectivity (JDBC)

An API used to communicate with relational databases.

Kill And Fill

Kill And Fill refers to a data operation where all the existing data in a table is deleted (kill) and then replaced with new data (fill).

Manifest

A file within a bundle that lists the data files in the bundle and their interdependencies.

Operational Data Store (ODS)

The corporate data repository that services Oracle Retail Central Office.

ORBO

Oracle Retail Back Office

ORCO

Oracle Retail Central Office

ORLT

Oracle Retail Labels and Tags

ORMPOS

Oracle Retail Mobile Point of Service

ORPOS

Oracle Retail Point of Service

ORRM

Oracle Retail Returns Management

ReSA

Oracle Retail Sales Audit

RMS

Oracle Retail Merchandising System

RPM

Oracle Retail Price Management

RTLog

Retail Transaction Log

Seed Data

Seed Data is defined as data that must be supplied by our customers in order for our applications to fully use all features and functions which the customer decided to enable.

SIM

Oracle Retail Store Inventory Management

Store Applications

Oracle Retail applications that run in the store environment. This includes:

- Oracle Retail Back Office
- Oracle Retail Point-of-Service
- Oracle Retail Mobile Point-of-Service
- Oracle Retail Strategic Store Solutions
- Oracle Retail Labels and Tags
- Oracle Retail Store Inventory Management
- Oracle Retail Central Office
- Oracle Retail Returns Management.

It must be noted that even though Oracle Retail Central Office runs in the corporate environment, it is classified as a store application.

Store Database

The data repository for store applications.

Strategic Store Solutions

The Oracle Retail business unit that assumes responsibility for applications running in the Store environment.

