Oracle® Retail POS Suite

Implementation Guide – Volume 1, Implementation Solutions

Release 13.4

E26984-01

December 2011

ORACLE®

Oracle Retail POS Suite Implementation Guide – Volume 1, Implementation Solutions, Release 13.4

E26984-01

Primary Author: Graham Fredrickson

Value-Added Reseller (VAR) Language

Oracle Retail VAR Applications

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

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

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

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

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

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

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

Contents

6 Backend System Administration and Configuration

14 Intra Store Data Distribution Infrastructure

15 Audit Logging

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle Retail POS Suite Implementation Guide – Volume 1, Implementation Solutions, Release 13.4

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

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

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

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

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

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

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

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

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

Preface

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.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired.

Related Documents

For more information, see the following Release 13.4 documentation sets or Oracle Retail Returns Management Release 2.4 documentation set:

- Oracle Retail Back Office documentation set
- Oracle Retail Central Office documentation set
- Oracle Retail Labels and Tags documentation set
- Oracle Retail Point-of-Service documentation set
- Oracle Retail POS Suite documentation set
- Oracle Retail Returns Management documentation set

Customer Support

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

- https://support.oracle.com

When contacting Customer Support, please provide the following:

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

Review Patch Documentation

If you are installing the application for the first time, you install either a base release (for example, 13.4) or a later patch release (for example, 13.4.1). If you are installing a software version other than the base release, be sure to read the documentation for each patch release (since the base release) before you begin installation. Patch documentation can contain critical information related to the base release 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 (with the exception of the Data Model which is only available with the release packaged code):

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.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Introduction

This guide includes information that is useful for configuring specific features in the Oracle Retail POS Suite.

This implementation guide addresses the following topics:

- Chapter 2, "Oracle Retail POS Suite Technical Architecture": Contains information about the Back Office, Central Office, Mobile Point-of-Service and Returns Management architecture.
- Chapter 3, "Oracle Retail Returns Management Overview": Introduction to Oracle Retail Returns Management.
- Chapter 4, "Oracle Retail Returns Management Functional Design and Overview": Addresses the functional aspects of Returns Management.
- Chapter 5, "Store Database": Describes the database used with Point-of-Service and how to interface with it. The chapter includes an example of writing code to store new data in the database.
- Chapter 6, "Backend System Administration and Configuration": Options for configuring Back Office, Central Office, Labels and Tags, Returns Management and Point-of-Service normally carried out by an administrator before the system goes into general use.
- Chapter 7, "Oracle Retail Returns Management Integration Methods and Communication Flow": Discusses the methods and the messages used by Returns Management to communicate between systems.
- Chapter 8, "Oracle Retail Point-of-Service to Oracle Retail Store Inventory Management Architecture": Describes the Point-of-Service to Store Inventory Management integration.
- Chapter 9, "Changing and Configuring Currencies": Steps for changing an existing base currency, or adding a new base currency.
- Chapter 10, "Configuring Multiple Printers for Labels and Tags": Provides configuration information for using multiple printers to print labels and tags.
- Chapter 11, "Returns Authorization": Provides an overview of the integration with Returns Management.
- Chapter 12, "Bill Pay": Describes the Bill Pay feature that enables retailers to accept bill payments from their customers and interface with their billing system to record the payments.
- Chapter 13, "Authorized Payment Foundation": Describes the Authorized Payment Foundation utility that provides an interface to third-party payment application providers.

- Chapter 14, "Intra Store Data Distribution Infrastructure": Describes the Intra Store Data Distribution infrastructure for POS Suite.
- Chapter 15, "Audit Logging": Provides information about audit logging for POS Suite.
- Chapter 16, "Oracle Retail Returns Management Customer Data Import": Describes the Returns Management customer import feature. This feature is a way for a retailer to import a large amount of pre-existing customer data into the data-store accessed by Returns Management.
- Chapter 17, "Centralized Customer": Describes the Centralized Customer feature for POS Suite.
- Chapter 18, "Automated E-Mail Messages": Describes the automated e-mail messages feature for Point-of-Service.
- Chapter 19, "Register Cash Notification": Describes the register cash notification feature for Point-of-Service that provides added security and allows stores to manage cash by register and till.
- Chapter 20, "Receipt Builder": Describes the receipt builder tool for Point-of-Service.
- Chapter 21, "Scan Sheet": Describes the scan sheet feature for Point-of-Service, including configuration.
- Chapter 22, "Serial Numbers": Provides an overview of how serial numbers are used in POS Suite, and configuration to use serial numbers.
- Chapter 23, "Oracle Retail Returns Management Exception File": Describes the Returns Management exception file. The exception file is created and maintained for use in detecting and preventing fraud at the point-of-return.

2

Oracle Retail POS Suite Technical Architecture

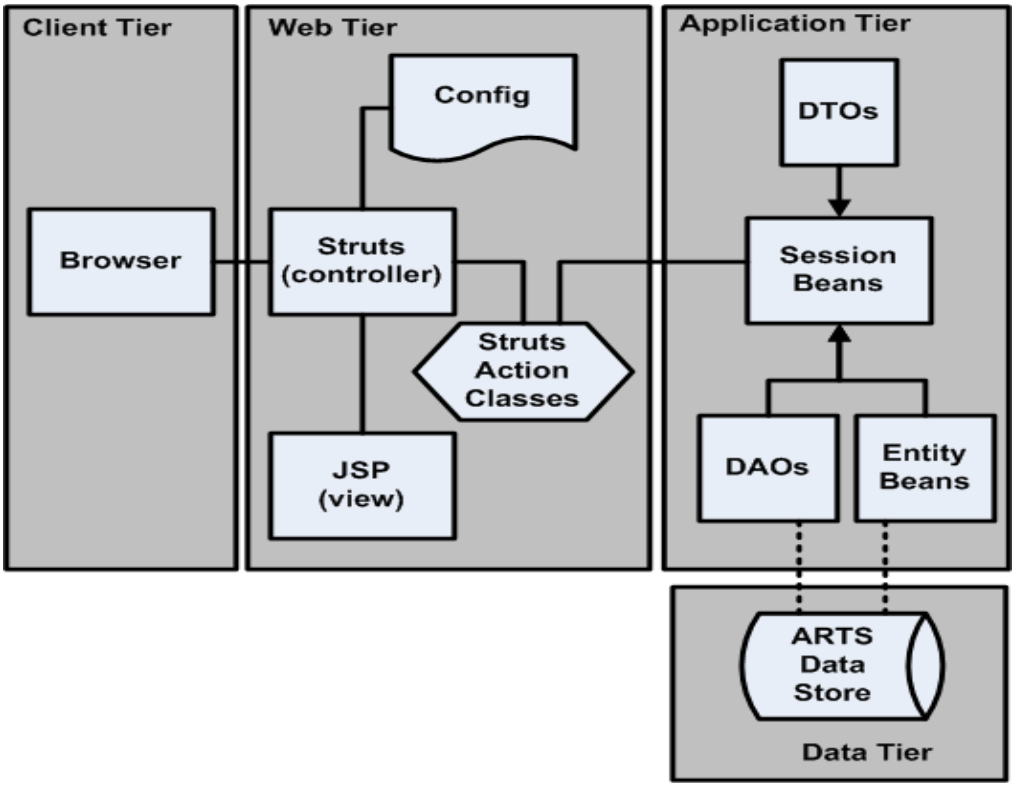
This chapter contains information about the Oracle Retail Back Office, Central Office, Point-of-Service, and Returns Management architecture.

Back Office and Central Office Tier Organization

The architecture of Back Office and Central Office uses client, middle, and data tiers. The client tier is a Web browser; the middle tier is deployed on an application server; and the data tier is a database deployed by the retailer.

The middle tier is organized in an MVC design pattern, also called a Model 2 pattern. This chapter focuses on the middle tier and the model, view, and controller layers that it is divided into.

Figure 2–1 High-Level Architecture



Client Tier

The client system uses a Web browser to display data and a GUI generated by the application. Any browser which supports JavaScript, DHTML, CSS, and cookies can be used. In practice, only a few popular browsers are tested.

Middle Tier

The middle tier of the application resides in a J2EE application server framework on a server machine. The middle tier implements the MVC pattern to separate data structure, data display, and user input.

Model

The model in an MVC pattern is responsible for storing the state of data and responding to requests to change that state which come from the controller. In Back Office and Central Office, this is handled by a set of Commerce Services, which encapsulates all of the business logic of the application. The Commerce Services talk to the database through a persistence layer of entity EJBs, using bean-managed persistence, or data access objects (DAO).

Commerce Services are components that have as their primary interface one or more session beans, possibly exposed as Web services, which contain the shared retail business logic. Commerce Services aggregate database tables into objects, combining sets of data into logical groupings. Commerce Services are organized by business logic categories rather than application functionality. These are services like Transaction, Store Directory, or Parameter that would be usable in any retail-centric application.

These services in turn make use of a persistence layer made up of entity beans or DAOs. Each Commerce Service talks to one or more entity beans or DAOs, which map the ARTS standard database schema. A data access object (DAO) is an object that provides an abstract interface to some type of database or persistence mechanism, providing some specific operations without exposing details of the database. Using the bean-managed persistence (BMP) pattern, each entity bean maps to a specific table in the schema, and knows how to read from and write to that table. A DAO maps to one or more tables that belong to the same logical unit. The DAO is responsible for reading or writing data to and from these tables. The Commerce Services thus insulates the rest of the application from changes to the database tables. Database changes can be handled through changes to a few entity beans.

The Commerce Services architecture is designed to facilitate changes without changing the product code. For example, you can:

- Replace a specific component's implementation. For example, the current Store Hierarchy service persists store hierarchy information to the ARTS database. If a customer site has that information in an LDAP server, the Store Hierarchy could be replaced with one that connected to the LDAP. The interface to the service need not change.
- Create a new service that wraps an existing service (keeping the interface and source code unchanged), but adds new fields. You might create My Customer Service, which uses the existing Customer Service for most of its information, but adds some specific data. All that you change is the links between the Application Manager and the Customer Service.

View

The view portion of the MVC pattern displays information to the user. In Back Office this is performed by a Web user interface organized using the Struts/Tiles framework from the open-source Apache Foundation. Using Tiles for page layout enables greater use of the user interface components to enhance the extensibility and customization of the user interface.

To make the view aware of its place in the application, the Struts Actions call into the Application Manager layer for all data updates, business logic, and data requests. Any code in the Struts Actions should be limited to formatting data for the Java server pages (JSPs) and organizing data for calls into the Application Manager layer.

JSPs deliver dynamic HTML content by combining HTML with Java language constructs defined through special tags. Back Office pages are divided into Tiles which provide navigation and page layout consistency.

Figure 2–2 Tiles in an Oracle Retail Application

Controller

The controller layer accepts user input and translates that input into calls to change data in the model layer, or change the display in the view layer. Controller functions are handled by Struts configuration files and Application Services.

Struts Configuration The application determines which modules to call, on an action request, based on the struts-config.xml file.There are several advantages to this approach:

- The entire logical flow of the application is in a hierarchical text (xml) file. This makes it easier to view and understand, especially with large applications.
- The page designer does not need to read Java code to understand the flow of the application.
- The Java developer does not need to recompile code when making flow changes.

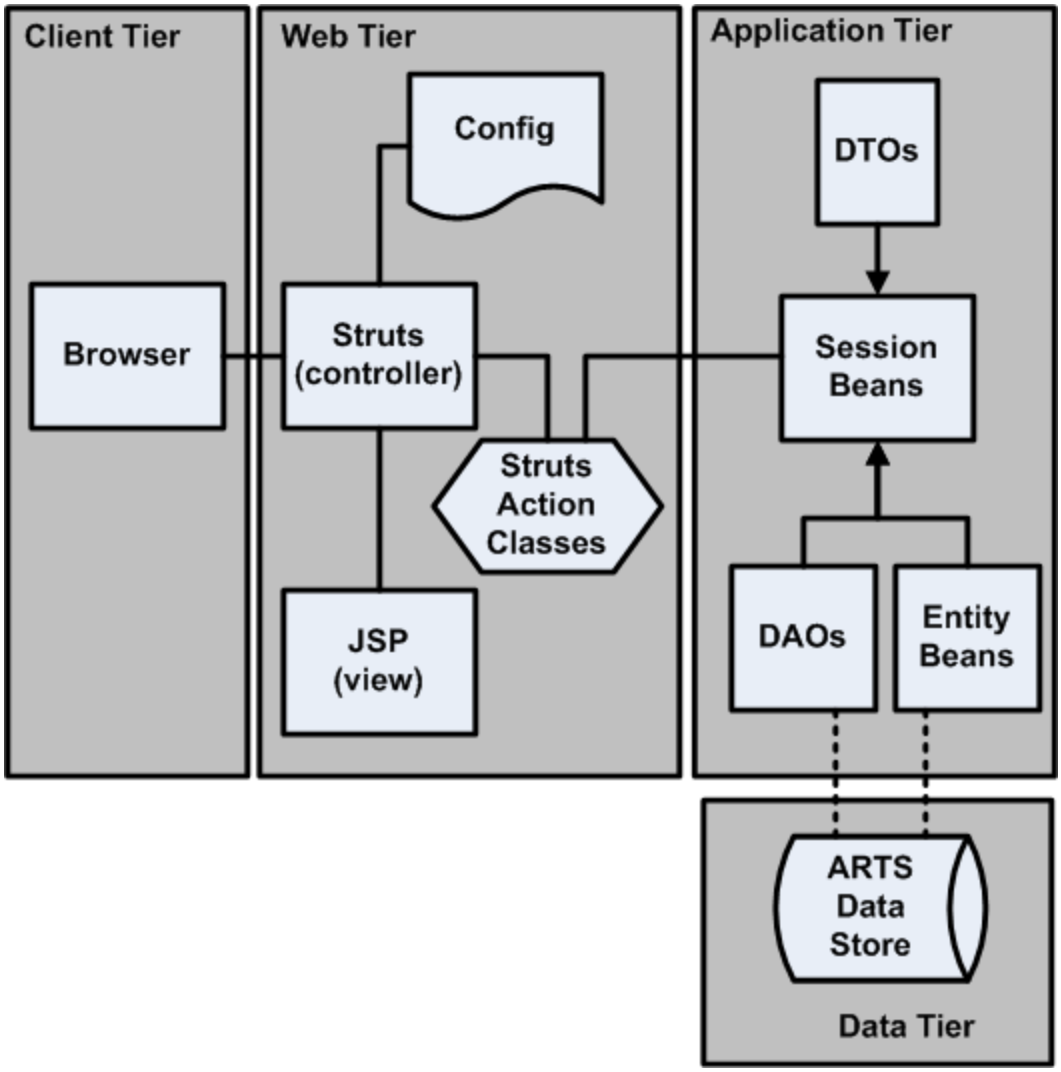
Struts reads the struts-config.xml once, at startup, and creates a mapping database (a listing of the relationships between objects) that is stored in memory to speed up performance.

Application Services The application services layer contains logical groupings of related functionality specific to the Back Office application components, such as Store Operations. Each grouping is called an application manager. These managers contain primarily application logic. Retail domain logic should be kept out of these managers and instead shared from the Commerce Services tier.

The application services use the Session Facade pattern; each Manager is a facade for one or more Commerce Services. A typical method in the Application Services layer aggregates several method calls from the Commerce Services layer, enabling the individual Commerce Services to remain decoupled from each other. This also strengthens the Web user interface tier and keeps the transaction and network overhead to a minimum.

For example, the logic for assembling and rendering a retail transaction into various output formats are handled by separate Commerce Services functions. However, the task of creating a PDF file is modeled in the EJournal Manager, which aggregates those separate Commerce Service functions into a single user transaction, thus decreasing network traffic and lowering maintenance costs.

Figure 2–3 Application Manager as Facade for Commerce Services



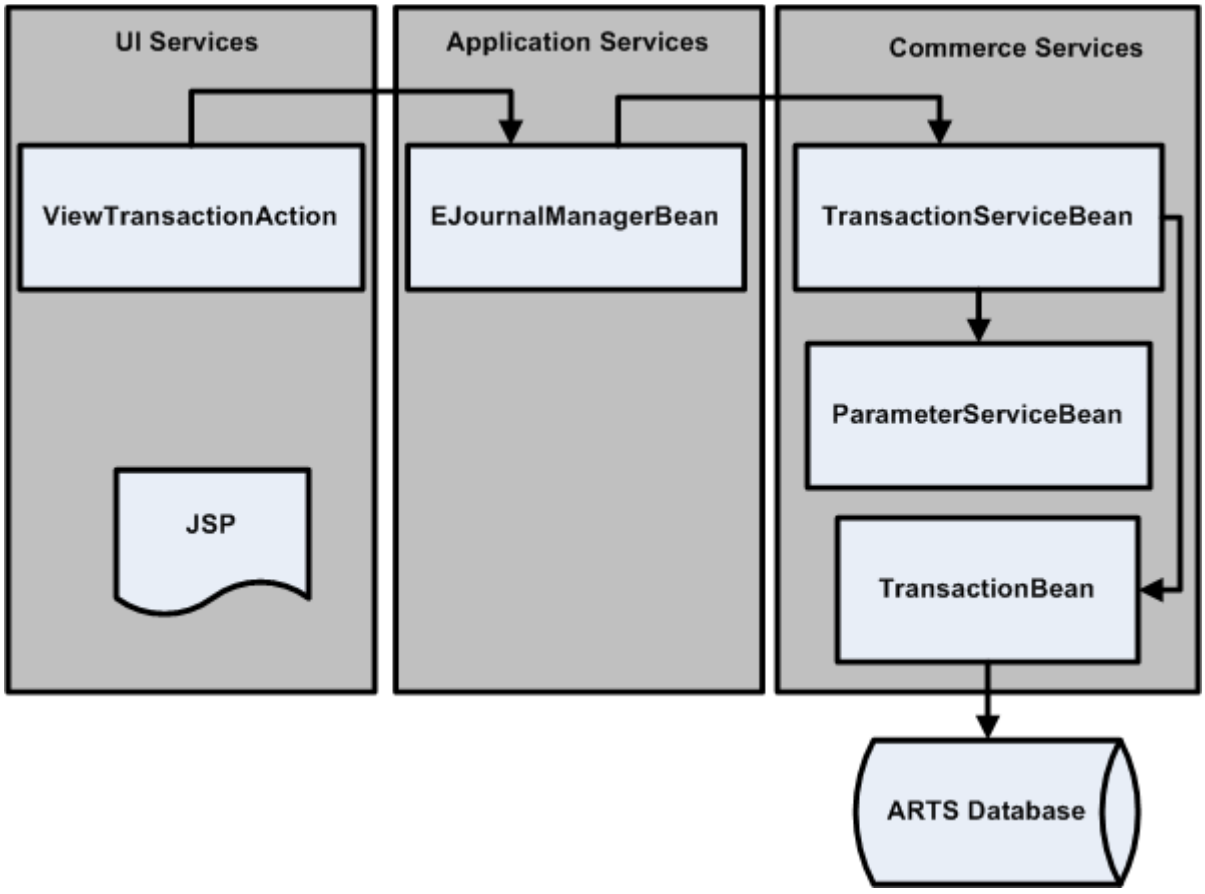
Data Tier

The Data Tier is represented by a database organized using the ARTS standard schema. Customer requirements determine the specific database selected for a deployment.

Dependencies in Application and Commerce Services

Figure 2–4 shows representative components Application Services and Commerce Services. Arrows show the dependencies among various components.

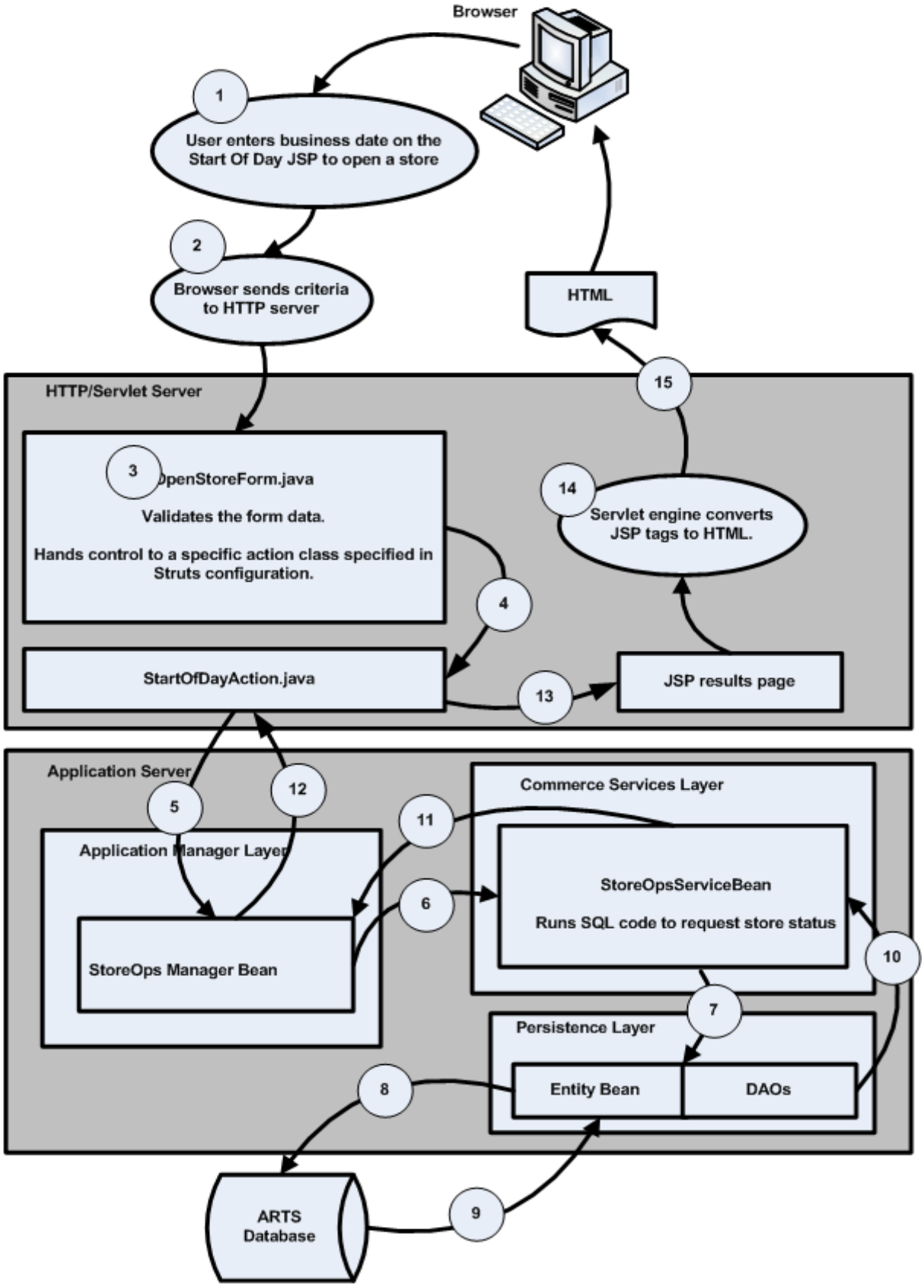
Figure 2–4 Dependencies in Back Office



Example of Operation

Figure 2–5 describes a trip through the Back Office architecture, starting from a user's request for specific information and following through until the system's response is returned to the user's browser.

Figure 2–5 Operation of Back Office



Point-of-Service Architecture

Retailers have an increasing demand for enterprise information and customer service capabilities at a variety of points of service, including the Internet, kiosks and handheld devices. The retail environment requires that new and existing applications can be changed quickly in order to support rapidly changing business requirements. Oracle Retail Platform and Retail Domain enable application developers to quickly build modifiable, scalable, and flexible applications to collect and deliver enterprise information to all points of service.

Figure 2–6 shows a high level view of the Oracle Retail architecture and components.

Figure 2–6 Oracle Retail Architecture

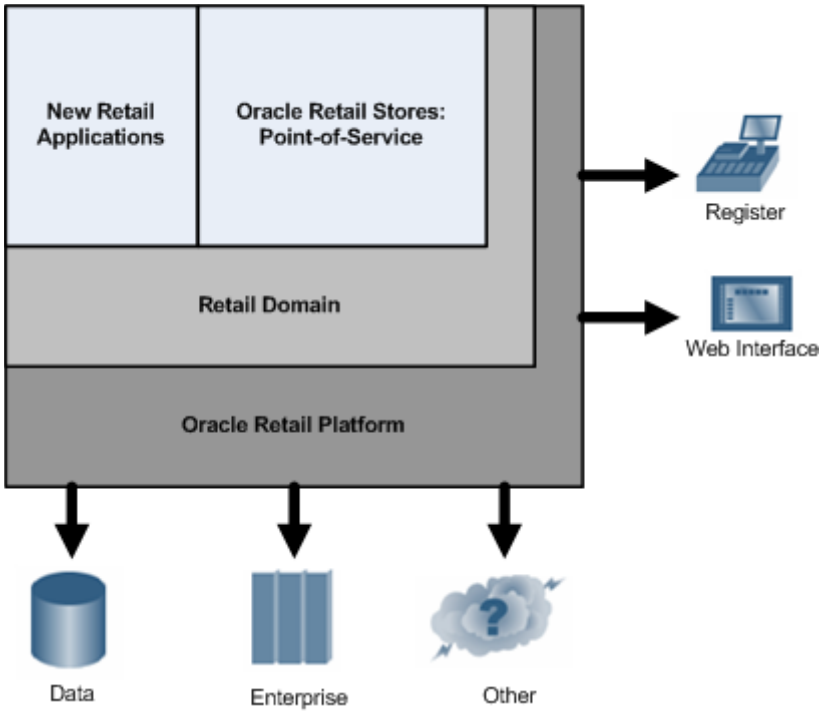


Table 2–1 describes the components in Figure 2–6.

Table 2–1 Oracle Retail Architecture Components

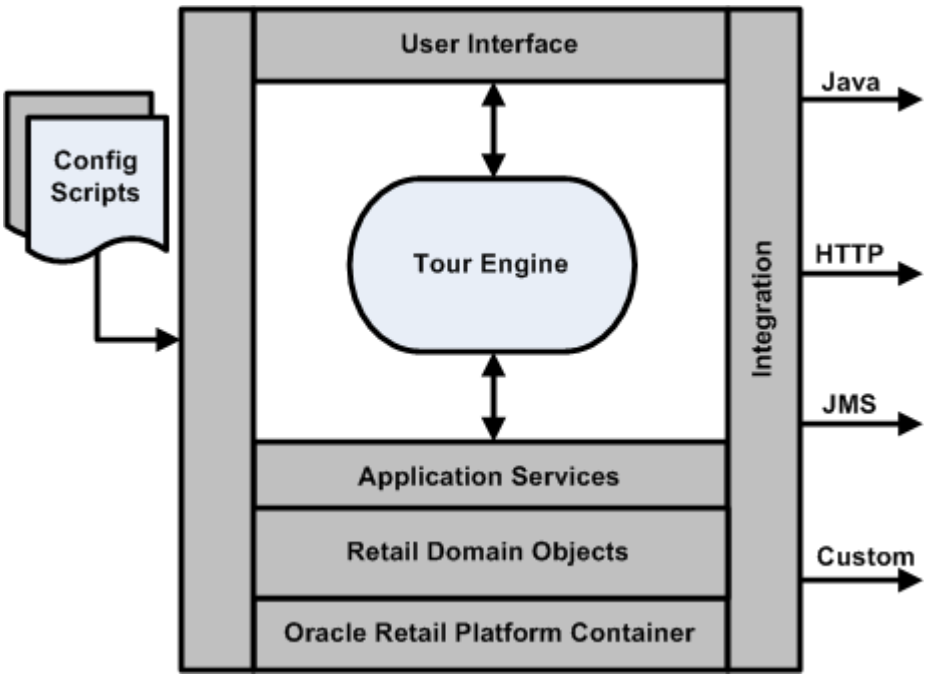
Component	Description
Oracle Retail Platform	Oracle Retail Platform provides services to all Oracle Retail applications. It contains the tour framework, UI framework, and Manager/Technician frameworks. Oracle Retail Platform is not retail-specific.
Retail Domain	Retail Domain implement business logic. Retail Domain defines data and behavior for retail applications.
Oracle Retail Applications	All Oracle Retail applications leverage the frameworks and services provided by Oracle Retail Platform and Commerce Services.
External Interfaces	Using frameworks and services, the applications are able to interface to other applications and resources.

Advantages of the Oracle Retail architecture include its object-oriented design and scalability. The system is designed to support existing systems and customer extensions. Oracle Retail Platform frameworks support integration by adhering to retail and technology standards. The multi-tier design of the architecture allows the application to support numerous types of infrastructure.

Oracle Retail Platform contains reusable, highly customizable components for building and integrating retail applications with user interfaces, devices, databases, legacy systems, and third-party applications. Oracle Retail Platform also contains integration points for communicating with external resources.

Figure 2–7 shows how the Tour engine controls the Point-of-Service system. This diagram is a more detailed view of the components that form the Retail Domain and Oracle Retail Platform tiers in Figure 2–6.

Figure 2–7 Point-of-Service Architecture Layers



Beginning with configuration of the UI and Managers/Technicians, events at the user interface are handled by the tour engine, which interacts with tour code (Application Services) and Managers/Technicians (foundation services that part of the oracle Retail platform layer) as necessary, capturing and modifying the data stored in Retail Domain objects. Any communication with an integration point is handled by the Oracle Retail Platform container.

Table 2–2 describes the layers of the Point-of-Service architecture.

Table 2–2 Point-of-Service Architecture Layers

Component	Description
Configuration	Application and system XML scripts configure the layers of the application.
User Interface	This layer provides client presentation and device interaction.
Tour Engine	This mechanism handles the workflow in the application. The tour engine is the controller for Point-of-Service.
Application Services	This layer provides application-specific business processes. A tour is an application service for Point-of-Service.
Retail Domain Objects	Pure retail-specific business objects that contain application data.
Oracle Retail Platform Container	This is an execution platform and application environment. The Tier Loader is the Oracle Retail Platform container for Point-of-Service. It contains the tour framework, UI framework, and Manager/Technician frameworks.
Integration	This layer provides an integration framework for building standard and custom interfaces using standard integration protocols.

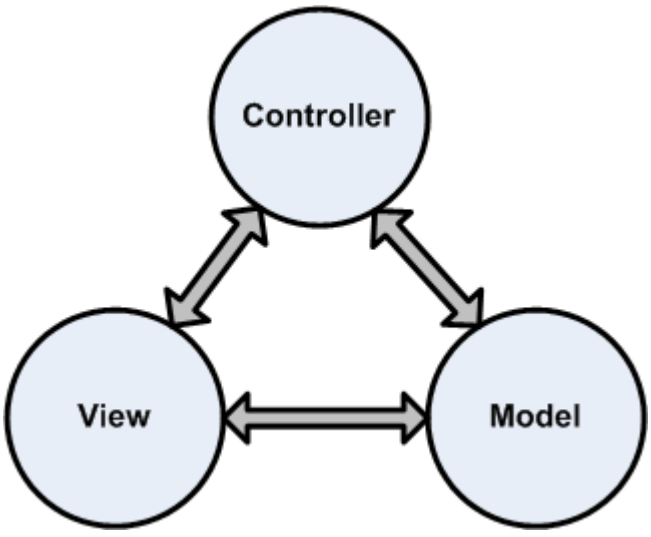
Design Patterns

Design patterns describe solutions to problems that occur repeatedly in object-oriented software development. A pattern is a repeatable, documented method that can be applied to a particular problem. This section describes four patterns used in the architecture of Point-of-Service: MVC, Factory, Command, and Singleton.

MVC Pattern

The MVC Pattern divides the functionality of an application into three layers: model, view, and controller. Different functionality is separated to manage the design of the application. A model represents business objects and the rules of how they are accessed and updated. The model informs views when data changes and contains methods for the views to determine its current state. A view displays the contents of a model to the user. It is responsible for how the data is presented. Views also forward user actions to the controller. A controller directs the actions within the application. The controller is responsible for interpreting user input and triggering the appropriate model actions. Figure 2–8 illustrates the MVC Pattern.

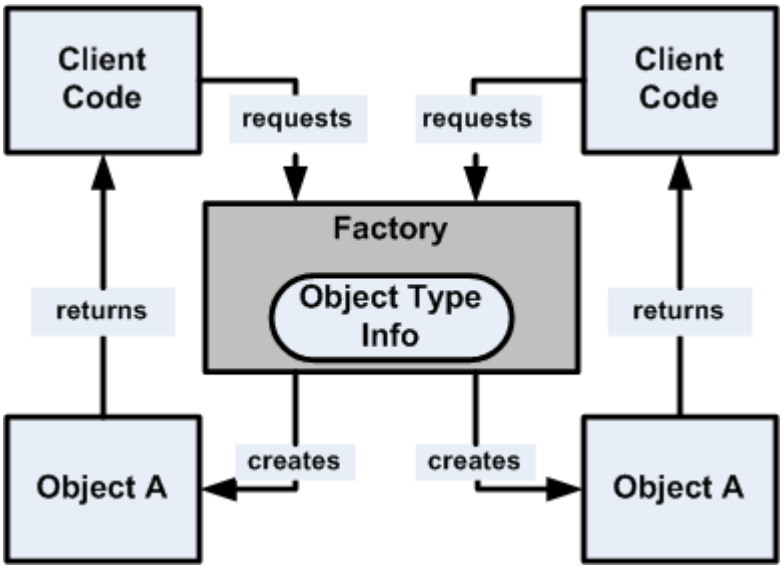
Figure 2–8 MVC Pattern



Factory Pattern

Another design pattern used in Point-of-Service code is the Factory pattern. The intent of the Factory pattern is to provide an interface for creating families of related or dependent objects without specifying their concrete classes. The application requests an object from the factory, and the factory keeps track of which object is used. Since the application does not know which concrete classes are used, those classes can be changed at the factory level without impacting the rest of the application. Figure 2–9 illustrates this pattern.

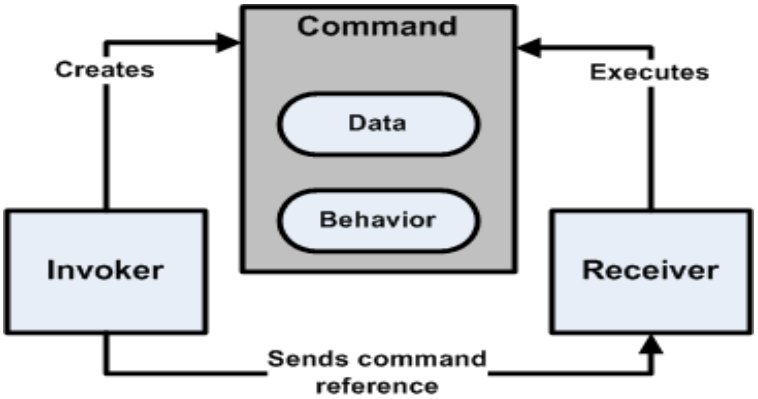
Figure 2–9 Factory Pattern



Command Pattern

Sometimes it is necessary to issue requests to objects without knowing anything about the operation being requested or the receiver of the request. The Command pattern encapsulates a request as an object. The design abstracts the receiver of the Command from the invoker. The command is issued by the invoker and executed on the receiver. Figure 2–10 illustrates the Command pattern. It is used in the design of the Manager/Technician framework.

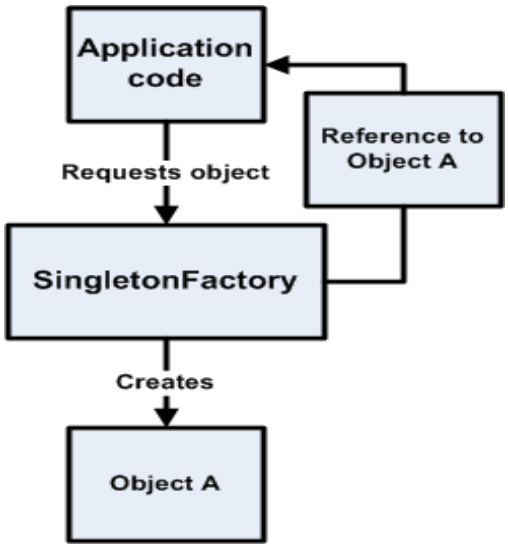
Figure 2–10 Command Pattern



Singleton Pattern

The Singleton pattern ensures a class only has one instance and provides a single, global point of access. It allows extensibility through subclassing. Singletons allow retailers to access the subclass without changing application code. If a system only needs one instance of a class across the system, and that instance needs to be accessible in many different parts of a system, making that class a Singleton controls both instantiation and access. Figure 2–11 illustrates the Singleton pattern:

Figure 2–11 Singleton Pattern



Returns Management Architecture

This section presents a concise description of the system's architecture. The system is considered a collection of run time behaviors, a set of software modules, and a member of a larger group of external systems and actors.

General Technologies and Frameworks

This section describes technologies and frameworks that are used by Returns Management. These assets are not unique to Returns Management but are key to its implementation.

Architectural Styles and Patterns

The following information describes the architectural styles and patterns of Returns Management and its component pieces.

Architectural Layers The architectural layers design style is not unique to Returns Management. It is a shared architecture across all of Oracle Retail's Web applications.

Figure 2–12 Oracle Retail Returns Management Architectural Layers

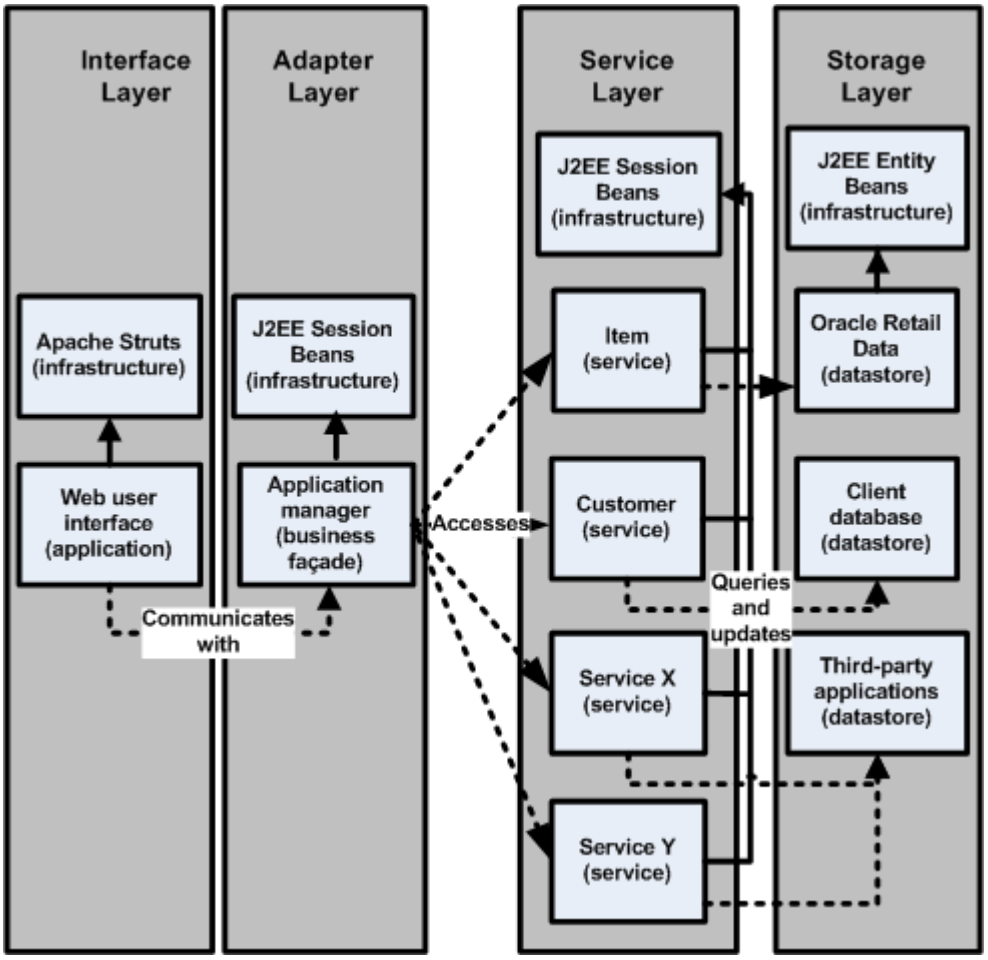


Figure 2–12 shows the break down of the user interface and business logic across the application. The design uses both a model-view-controller pattern as well as a façade pattern to hide the implementation of business logic. The façade not only applies to the user interface, but to the other pieces of business logic as well.

As with all object construction, the goal is to reduce the dependency between the objects so that code might be more freely modified without adversely affecting other parts of the application.

Apache Struts is used at the graphical user interface layer to provide both a clear separation of the controller, view, and model as well as providing a well known technology for ease of extension. At the façade layer, an application manager is implemented using J2EE session beans to provide a coarse-grained view of business logic to the user interface. Each manager communicates directly with one or more services located in the service layer that provide fine-grained business operations. These service beans are also implemented using J2EE session beans. Each service bean can then communicate with other services or down to persistent storage in the data layer. By abstracting the storage away from the other layers, this not only enables the design to leverage J2EE entity beans but also allows for disparate storage mediums for integrating with third party data stores.

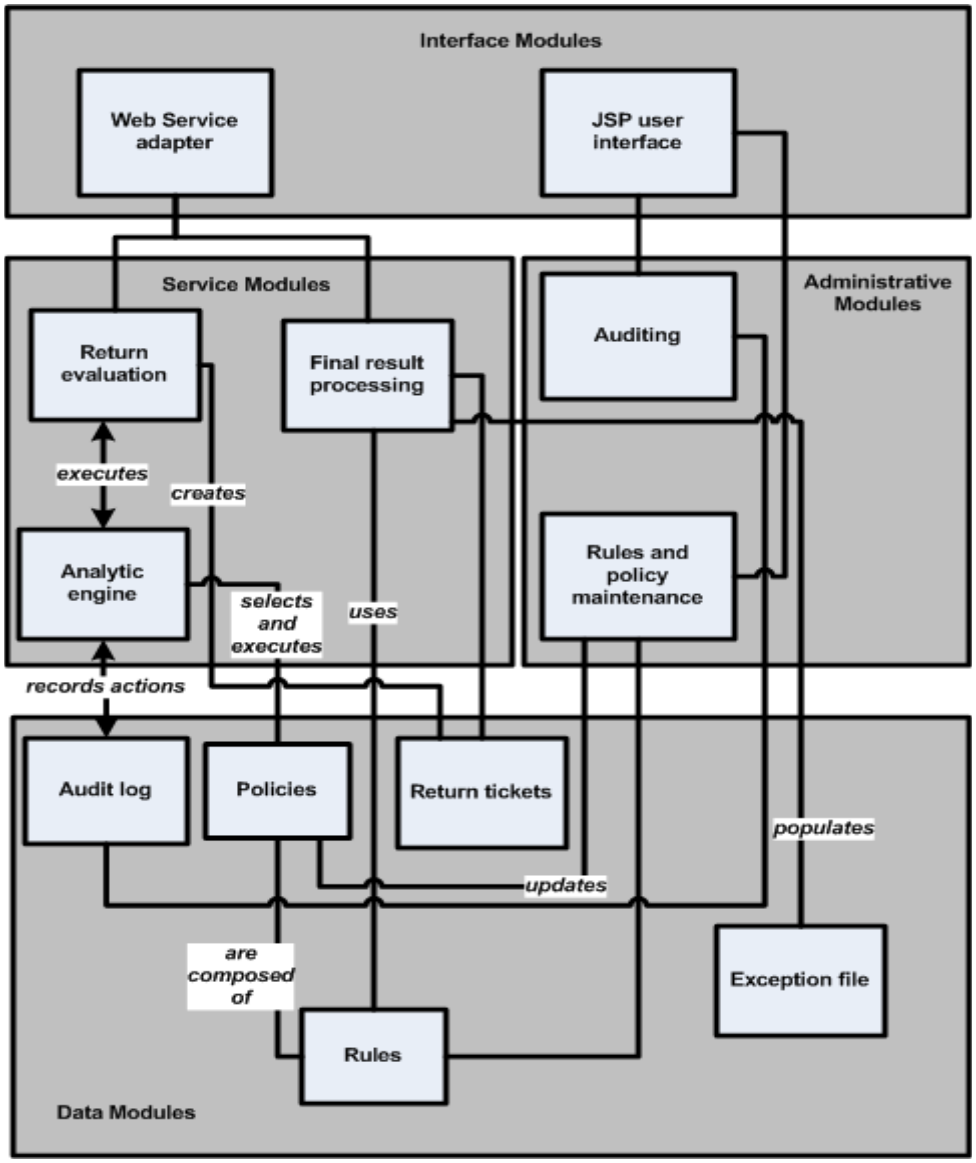
Conceptual Modules

Figure 2–13 is a conceptual view of the modules that make up the Returns Management service. Here, a module corresponds to code that provides a discrete piece of functionality. For example, the Auditing module corresponds to the auditing function.

For illustrative purposes, the code has been split into four broad types:

- Interface modules
- Service modules
- Administrative modules
- Data modules

Figure 2–13 Oracle Retail Returns Management Conceptual Modules



Interface modules

Functionality to interact with an outside actor, such as the user or the point-of-return. These modules include:

- **Web Service Adapter**

  The point-of-return is expected to talk to Returns Management using a Web service. An adapter is provided to enable this functionality out-of-the-box.

- **JSP UI**

  The user interface is HTML-based, powered by JavaServer Pages (JSP) and Struts.

Service modules

Functionality that provides the core services offered by Returns Management. These modules include:

- **Return Evaluation**

  This is where the initial "Is this item returnable?" question is asked.

- **Final Result Processing**

  This is where Returns Management consumes results to maintain historical data.

- **Analytic Engine**

  The decision engine that evaluates Returns Management rules.

Administrative modules

Functionality to administer, monitor, and examine Returns Management behavior and data. These modules include:

- **Auditing**

  Provides the capability to examine the steps that went into a return decision.

- **Rules and Policy Maintenance**

  Enables the user to add, modify, or delete policies and the rules which comprise them.

Data modules

Functionality tied to persistent storage. These modules include:

- **Audit Log**

  The steps recorded by the engine as it processes a policy.

- **Policies**

  Collections of rules that are bound to certain conditions, for example, some policies apply only to receipted items.

- **Rules**

  Each rule evaluates a return-related question, for example, "How many returns has this customer attempted in the past week?" Rules are responsible for indicating to the point-of-return whether an item is returnable or not.

- **Return Tickets**

  Returns Management stores information about each return request for later manipulation by Final Result Processing.

- **Exception File**

  Records that store information about exceptional behavior, which are created when a customer has exhibited behavior the retailer wants to track.

Enabling Technologies

The following are example of enabling technologies in Returns Management.

JEE

Returns Management is built using the technologies of the Java Enterprise Edition stack.

Struts

Returns Management uses the Apache Struts project to present its Java Server Pages in a J2EE compliant container. For more information, go to
http://struts.apache.org/

Axis

Returns Management uses Apache Axis to provide a container-neutral way of presenting Web services. For more information, go to
http://ws.apache.org/axis/

Web-Based User Interface

The user interface for Returns Management can be divided into two classes of components:

- JSPs and Action Classes
- The Returns Manager

Figure 2–14 Oracle Retail Returns Management Web-based User Interface

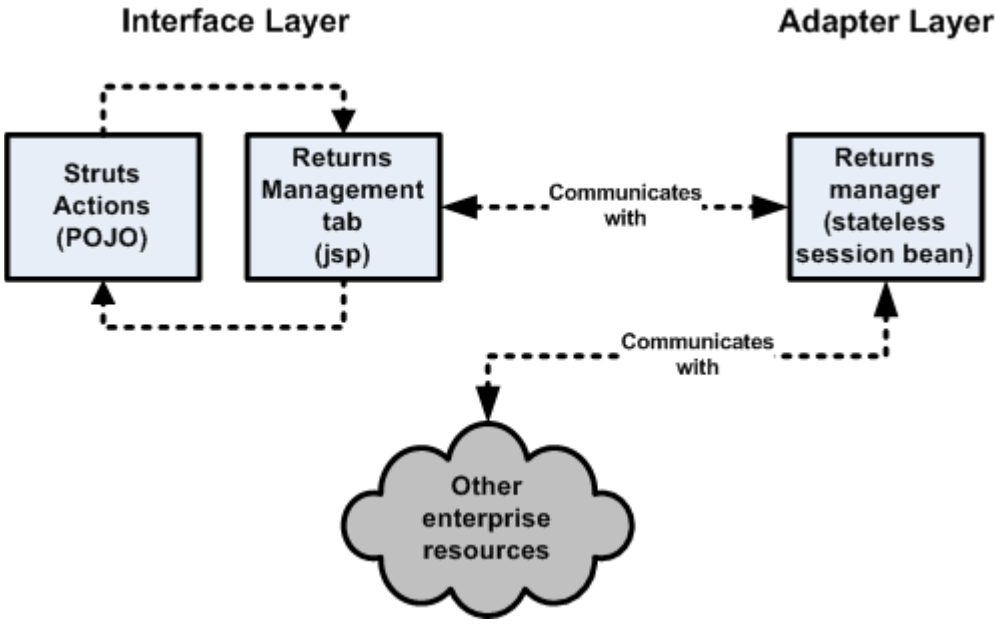


Figure 2–14 shows the general Web user interface diagram as it relates to Returns Management in particular. The interface layer is composed of the JSPs attached to the Returns Management as well as their backing action classes. These actions attempt to provide user interface-level functionality such as flow control and rudimentary data checking. The class at the façade layer, the Returns Manager, exposes numerous coarse-grained methods to enable the Action classes to retrieve key performance indicators (KPIs)—also known as return activities, search the exception file, and other administrative tasks. The Returns Manager then communicates to whatever resources it needs to provide the necessary information to the interface layer. Since all of this work is hidden behind the façade, the Returns Manager has flexibility in deciding how to perform a certain task with minimal impact to the client classes in the interface layer.

Physical Module View

The conceptual module view divided the system into modules based off of the functionality provided. The physical module view divides the modules along the notion of module type. For instance, rather than showing policy maintenance as a separate service, policy maintenance is included in the larger Returns Service.

Figure 2–15 Oracle Retail Returns Management Physical Module View

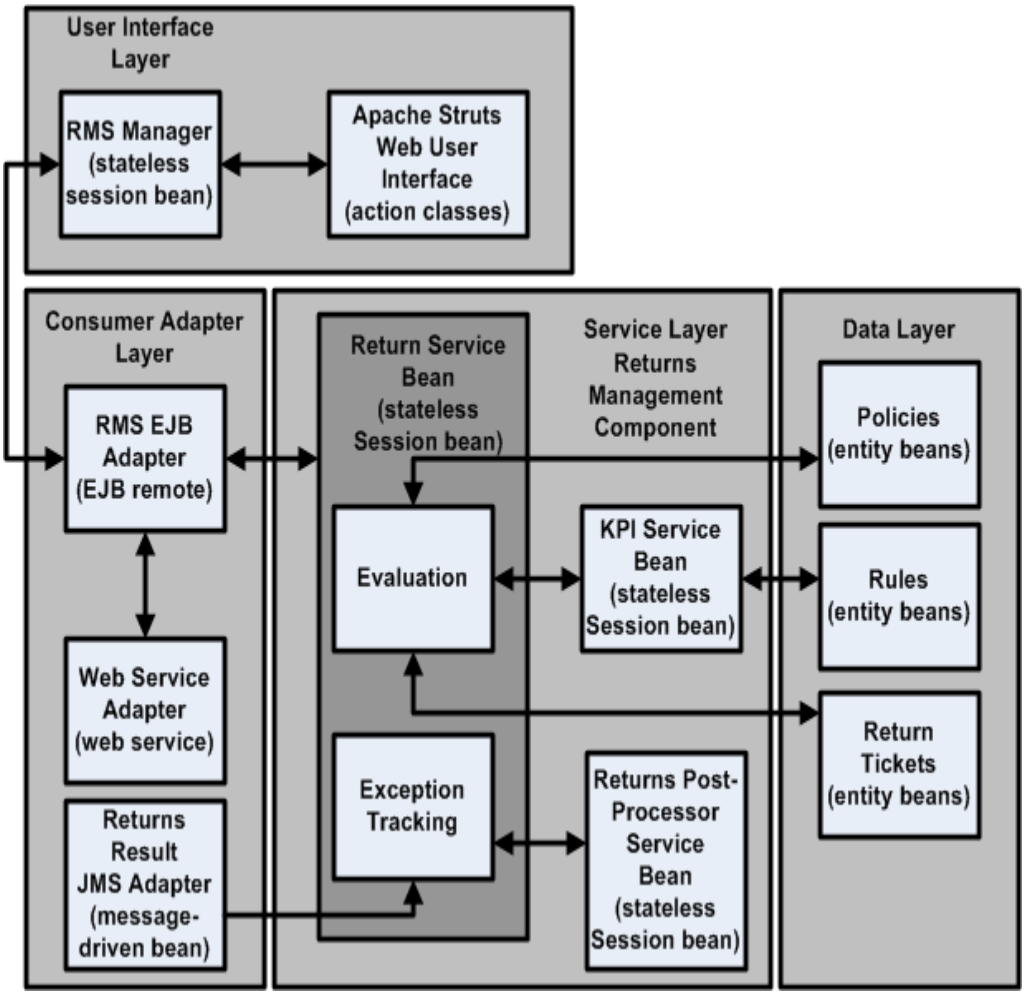


Figure 2–15 broadly divides the modules into four groups:

- User Interface Layer: responsible for the Web-based user interface.
- Consumer Adapter Layer: responsible for communication with Returns Management.
- Service Layer: provides the heavy lifting of Returns Management functionality.
- Data Layer: provides access to persistent storage.

User Interface Layer

As mentioned previously, the Returns Management user interface is a Web-based system implemented using Struts and Tiles. The presentation layer consists of a large number of JSPs, forms, actions, and other artifacts of the Struts system. Behind this presentation layer resides the Returns Manager façade, which provides the Struts actions with access to the business logic of the application.

In the Model-View-Controller paradigm, Struts provides all three pieces:

- Actions provide the model.
- JSPs are the view.
- Struts classes and configuration files provide the controller.

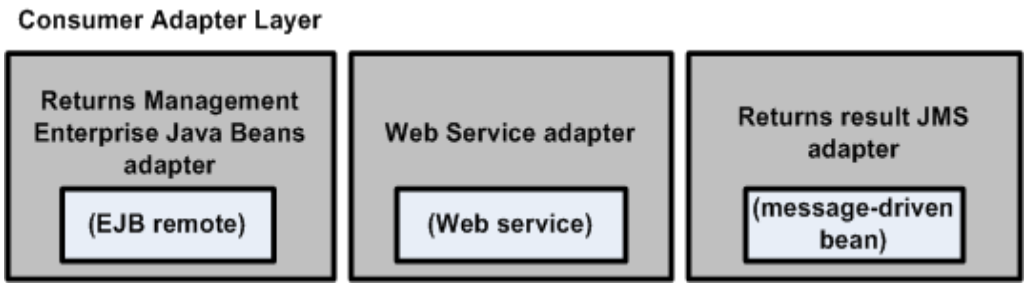
However, the action classes are not the model used by Returns Management. The action classes exist primarily to marshall data from the presentation layer down to the business logic, and to provide coarse-grained flow control over a business process. It is important to realize that the real model of Returns Management has little to do with the action classes. The real model is implemented behind the Manager façade in the service and data layers.

JavaServer Pages (JSPs) are text files which correspond to the normal JavaServer Page formatting restrictions. Actions and forms are normal Java classes while the Manager is implemented as a stateless session bean.

Consumer Adapter Layer

The consumer adapter layer provides the different interfaces into the Returns Management system. This layer is specifically split from the service layer in the design to decouple the interface of communication from the implementation classes. Therefore, regardless of underlying changes in how Returns Management is implemented, the interface expected by clients can remain static. The separation provides a well-defined contract with which service consumers can interact, and enables future custom adapters to be integrated with minimal effort.

Figure 2–16 Oracle Retail Returns Management Consumer Adapter Layer



There are three ways to communicate with the Returns Management system:

- Return Request and Return Response are exposed using a Web Services interface.
- Return Results are collected using a JMS queue.
- The EJB remote interface, which enables arbitrary methods to be invoked on the service bean.

For more information about communicating with Returns Management, see Chapter 7, "Oracle Retail Returns Management Integration Methods and Communication Flow".

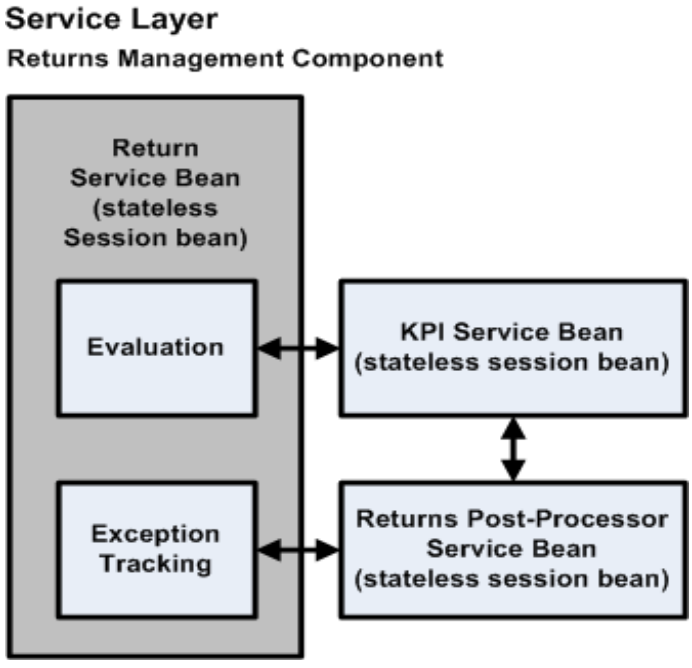
The first two interfaces mentioned enable flexible client implementations while providing clear interfaces for interaction. The EJB remote interface is available when some system needs to operate with Returns Management outside of the normal API-type transactions, for example, the Returns Manager makes liberal use of the remote interface for inquiring and maintaining Returns Management data.

Service Layer

The term service is used in two different ways when describing Returns Management.

- The first way is the abilities provided by Returns Management in its role as a service in a service-oriented architecture. These abilities are restricted to the two interfaces of evaluation (return request) and exception tracking (return result).
- The second way is used to describe the interoperable commerce services that form the core of Returns Management functionality. These are the services that live in the service layer. These services are generally not client accessible. They provide discrete business functions available to other services and Application Managers living in the façade layer.

Figure 2–17 Oracle Retail Returns Management Service Layer



When describing Returns Management in terms of service-oriented architecture, the services provided by Returns Management are implemented primarily in one class, the Return Service Bean. These services are exposed in the Consumer Adapter Layer for access from client routines.

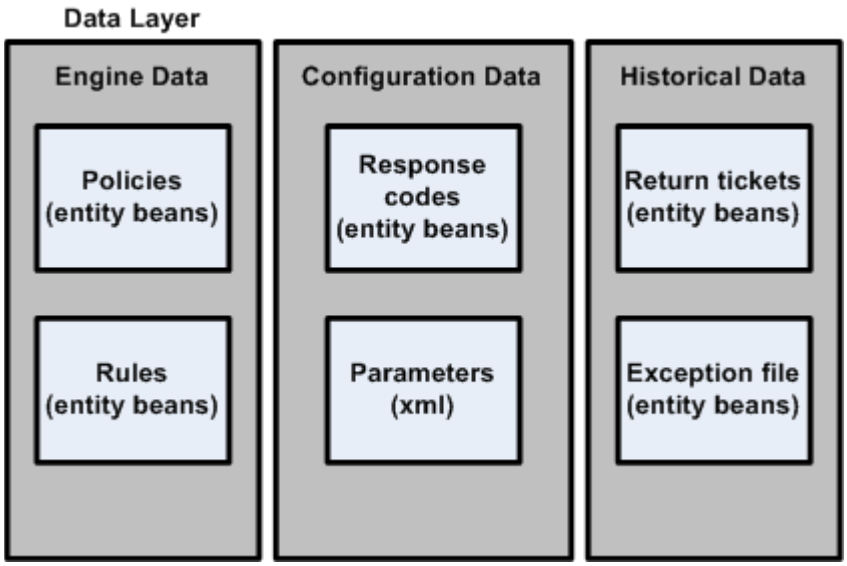
When describing Returns Management in terms of its commerce service modules, then not only would the return service bean be included but also the modules which exist to provide support to Returns Management, such as the KPI Service and the Returns Post-Processor Service.

Data Layer

For purposes of discussion, this document splits the data used by Returns Management into three types:

- Engine data used to determine returnability.
- Configuration data used to control behavior.
- Historical data used to record information.

Figure 2–18 Oracle Retail Returns Management Data Layer



These types of data are generally stored in an RDBMS and accessed using an entity bean layer. Each general type is covered briefly in the following sections.

Engine Data: Policies, Rules, And Return Activities Returns Management uses a decision engine to codify and enforce returns policies. The decision engine operates on sets of rules, which are collected into policies. The rules operate on a set of facts that the engine supplies at run time. These facts are evaluated by the rules, which eventually return an answer back from the engine. See "Determining Return Policies" in the *Oracle Retail Returns Management User Guide*.

Figure 2–19 Oracle Retail Returns Management Policies and Rules

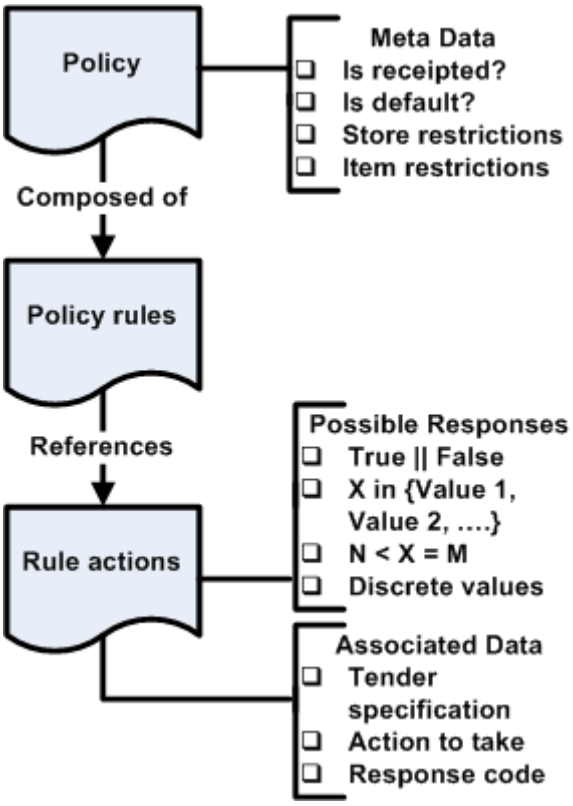


Figure 2–19 shows how policies, rules, and rule actions are conceptually related.

A **policy** is composed of one or more rules. Each policy has associated metadata that enables the service layer to choose the most appropriate policy for the current item in question.

A **rule** is configured to ask a certain question about the current item or customer in question. Each rule references several possible rule actions.

A **rule action** has two distinct functions:

- Identify an answer to a question. Each rule action corresponds to a value returned from the rule's question.
- Tell the analytic engine (for example, the service layer) what action to take in response to this answer, for example, whether to continue or stop policy evaluation and determine what response to return to the client.

Configuration Data Returns Management has a list of valid response codes that can be returned to a client. This list is maintained in the data store and is accessed using entity beans. Returns Management also maintains a list of receipt messages which are accessed in a similar fashion.

Following an established pattern in Oracle Retail products, Returns Management uses an XML parameter file to maintain a list of configuration options and choices that can be modified for a particular customer deployment. These parameters control a variety of behaviors as well as providing a place for some of the data used during processing (for instance, the list of acceptable tenders).

Note: For more information on specific parameters, see the *Oracle Retail POS Suite Configuration Guide*.

Historical Data Returns Management maintains a set of historical data. To record a particular decision during the evaluation phase, Returns Management creates a return ticket that records what item is being returned, who wants to return the item, and the Returns Management decision about the returnability of an item. This return ticket is later updated during the return result process to reflect what was actually returned at the point-of-return.

An exception file that counts up the total number of times an instance of a tracked behavior occurs, for example, a customer with a non-receipted return or a customer with a tender override, is maintained by Returns Management during the return result phase.

The following are grouped in the exception file:

- Customer exceptions, such as line items that reflect customer return activities being violated.
- Customer exception counts and freeze dates.
- Customer Service Overrides, that is, a count of overrides, per day, per customer.

For more information, see the following:

- "Selecting Customer Exceptions to Track" in the *Oracle Retail Returns Management User Guide*.
- Chapter 23, "Oracle Retail Returns Management Exception File".

Messaging

This section describes the interface of the two main services provided by Returns Management:

- Evaluation of the return request
- Processing of the return result

These services are expected to be invoked from an external source, usually the point-of-return.

In order to provide language neutrality, these two services are accessed in a stateless fashion using XML documents. Return request is a synchronous message, that is, the invoker is expected to wait on a return response message. Return result is an asynchronous message that can be invoked in a fire-and-forget fashion.

Although both services are exposed using a Web service interface, a message-driven bean exists to collect return result messages asynchronously as a best practice. More details about messaging are provided in Chapter 7, "Oracle Retail Returns Management Integration Methods and Communication Flow".

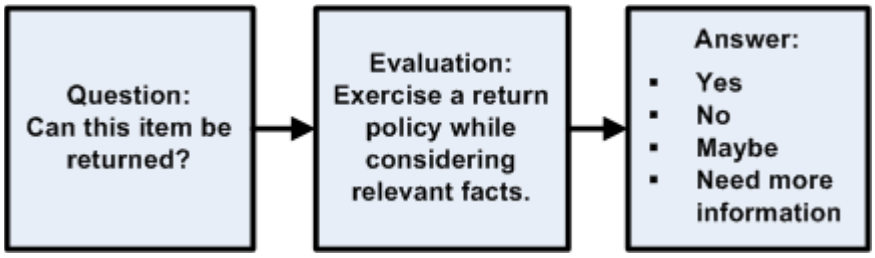
3

Oracle Retail Returns Management Overview

Oracle Retail Returns Management is a centralized system designed to monitor and control the return of retail merchandise. Control is provided through a flexible set of rules that determine if a particular item is returnable. Monitoring is provided through pattern watches, enabling a retailer to uncover unusual return patterns indicative of fraud, poor product quality, and so on.

At its most basic, Returns Management enables you to centralize the knowledge and decision making of what is and what is not returnable.

Figure 3–1 Oracle Retail Returns Management Decisions Process



Returns Management is packaged as a standalone product.

Returns Management provides:

- A flexible and configurable set of rules
- The ability to collect differing rules into multiple policies
- The ability to assign different policies to different situations (for instance, one policy might apply to receipted items, another policy might apply to non-receipted items)
- A decision engine to initiate the policies
- A defined application-program interface (API) for evaluation of returnability
- A defined API for post-return information gathering
- A Web-based user interface for administration

Concept of a Return in Returns Management

Occasionally, a customer might buy an item from a retailer and then decide that they no longer want the item. This could be for any number of reasons:

- Dissatisfaction with quality
- Finding a better price somewhere else
- Buying the wrong size

Most retailers allow customers to return items they have purchased under certain conditions. Conditions might include that the item was bought within the last 90 days, that the item is in an unopened state, or that the customer has a receipt. Additionally, a retailer might decide to charge a restocking fee, issue a return merchandise authorization (RMA) and a call tag, provide a discount on the customer's next purchase (in case of a quality problem), or other actions based around the return. Finally, returns can happen at different places in a retailer, such as at a point-of-sale, a separate returns desk, or at a remote call center.

With the act of returning, there are several steps that a retailer must go through.

- A retailer must determine what merchandise is being returned.
- A retailer must decide if the merchandise is returnable.
- A retailer must record that the item was returned, which affects financial and inventory calculations.
- Once the retailer accepts a return, the retailer must physically move the item to some place (such as placing it in a returns cage, or issuing pickup instructions to a carrier in the case of a remote call center). Afterwards, the item might undergo further actions, such as being returned to the vendor or destroyed.

Of these many aspects of the return process, Returns Management focuses mainly on the conditions for return, sometimes referred to in this document as returnability. Returnability is determined by the rules and policies configured in Returns Management. Returns Management can associate metadata with return policies so Returns Management can decide to use different policies in different situations. Each policy has its own set of rules which define returnability for that situation, for example, non-receipted returns might be more restrictive than receipted returns.

A policy is composed of one or more rules. Each policy has associated metadata that enables the service layer to choose the most appropriate policy for the current item in question.

Returns Management does not prescribe for the point-of-return, what happens before or after a customer initiates a return, or the financial and inventory ramifications of the returns process. Furthermore, Returns Management isolates itself from the majority of data found in the retail enterprise and restricts itself to knowing a prescribed set of facts. It is this set of facts that Returns Management uses when evaluating a policy and determining if a product is returnable.

Returns Management has been isolated to this degree in order to make Returns Management applicable to a wide variety of situations. As long as a point-of-return can communicate with Returns Management, it is immaterial where that point-of-return is located and what kind of point-of-return it is (register, returns desk, and so forth). The point-of-return provides the majority of the data that Returns Management needs to make its decision, so Returns Management is shielded from the format of transaction data in a retail enterprise.

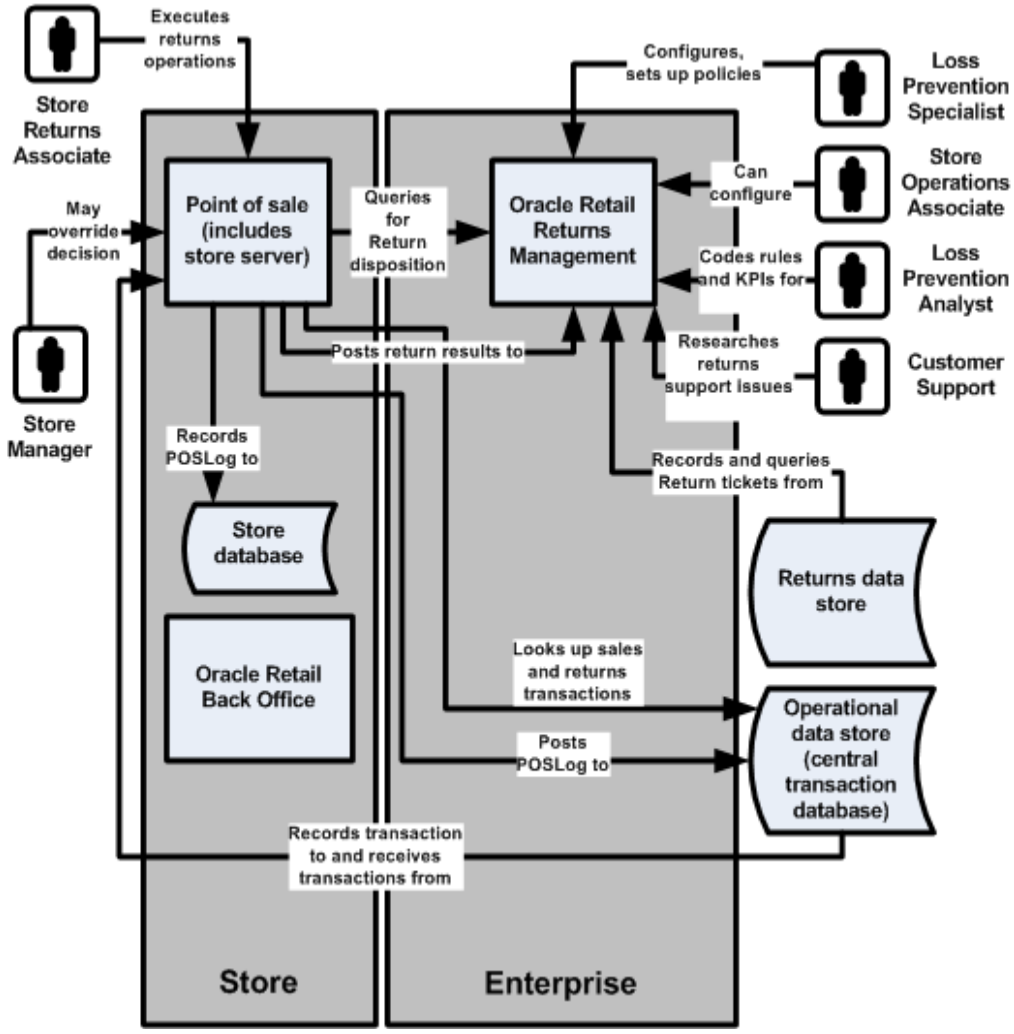
By configuring the rules and policies in Returns Management, the retailer can enforce the same return policies across the enterprise. The return policies can be centrally administered. Since the policies are not compiled code, they can be quickly updated. Finally, Returns Management records the steps it makes for each decision, allowing a customer to ask exactly why a return was accepted or declined.

As stated previously, Returns Management focuses mainly on the conditions for return. The other main focus of Returns Management is that it keeps a record of what was actually returned to the enterprise. This information is rolled up in both the return ticket data (there is a ticket for each return, each ticket having one or more line items corresponding to items on the return) as well as a list of exception activity. The exception file records unusual activity that might be fraudulent. Using these records, Returns Management provides decision support to the enterprise. The retail enterprise can monitor these records to determine the volume of returns, the type of items being returned, and patterns of fraudulent behavior.

Context Model

Figure 3–2 identifies how Returns Management exists with other existing Oracle Retail products. Also included in the context are the actors mentioned in Table 3–1, " Oracle Retail Returns Management Actors".

Figure 3–2 Oracle Retail Returns Management Context Model



Oracle Retail Returns Management Actors

Table 3–1 lists the actors that Returns Management expects to interact with, and their interactions. Although most of the actors are users, some items such as the point-of-return are expected to interact with Returns Management without direct human intervention.

Table 3–1 Oracle Retail Returns Management Actors

	STORE				CORPORATE			
	Sales Associate	**Point-of-return**	**Store Manager**		**Business analyst**	**Customer Service Rep**	**Loss Prevention Specialist**	**Software Developer**
Request return	X	NA	NA	NA	NA	X	NA	NA
Update return information	NA	X	NA	NA	NA	NA	NA	NA
Develop return policies	NA	NA	NA	NA	X	NA	NA	X
Monitor exception behaviors	NA	NA	X	NA	NA	NA	X	NA
Monitor what is being returned	NA	NA	X	NA	X	NA	X	NA
Audit a specific returns decision	NA	NA	NA	NA	X	X	NA	NA

Tax Responsibility in Oracle Retail Returns Management

Returns Management evaluates data provided from the point-of-return as well as centrally stored historical data and provides a recommendation to the point-of-return for the handling of a potential return. Because Returns Management is not transactional in nature, Returns Management has no tax responsibility. All of the tax responsibility belongs to the point-of-return when the return transaction is created and processed.

Note: Returns Management operates using a single default currency. If operations require using multiple currencies, it is the responsibility of the point-of-return to convert from any other currencies to the single default currency being used by the system. Returns Management does not provide services for the conversion of currency from one form to another.

4

Oracle Retail Returns Management Functional Design and Overview

This chapter addresses the functional aspects of Returns Management and provides the following:

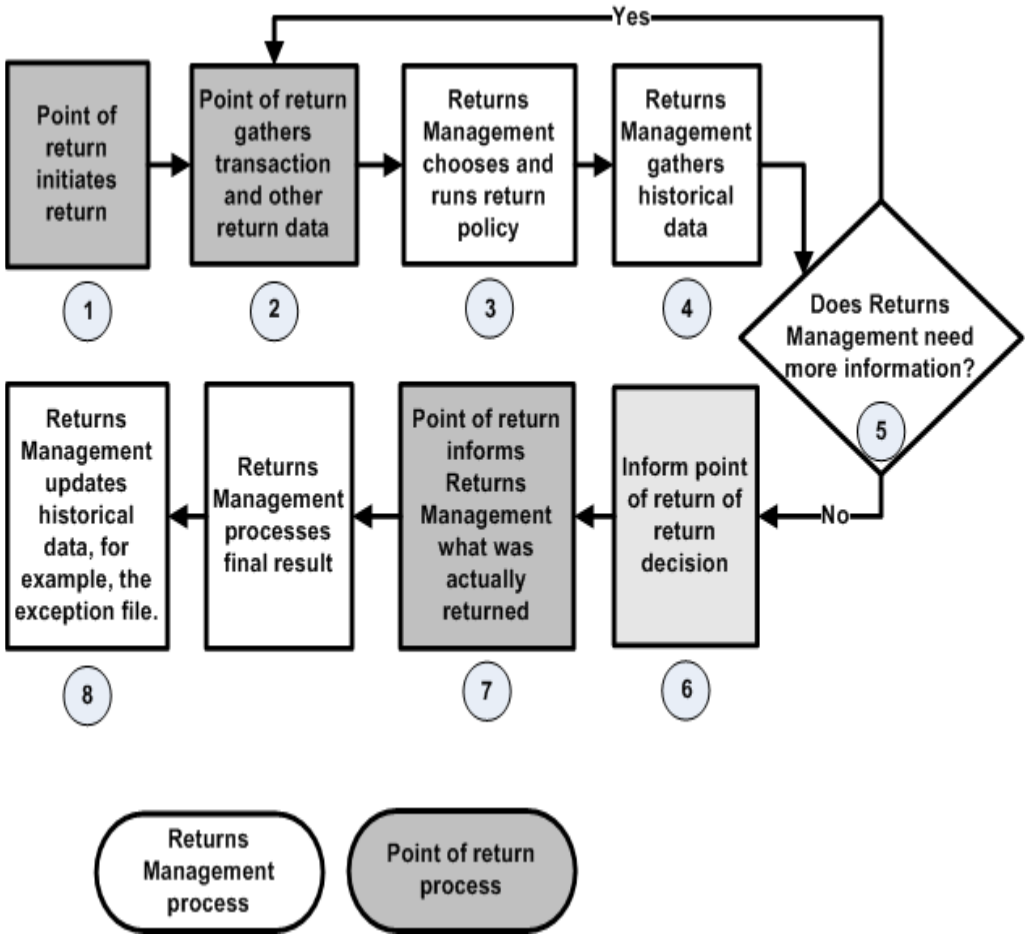
- Conceptual Service Flow
- Conceptual Data Flow
- Functional Assumptions
- Functional Overviews

Conceptual Service Flow

Figure 4–1 illustrates the steps in a typical Returns Management session, including which steps are initiated by Returns Management and which steps are initiated by the point-of-return (the point at which a return is initiated, for example, a cashier at a retailer).

Note: This flowchart is a simplified representation of the service flow and does not attempt to explain the technologies used to implement Returns Management.

Figure 4–1 Oracle Retail Returns Management Conceptual Service Flow



The following sequence is a typical Returns Management round-trip session:

1. A point-of-return initiates a merchandise return.
2. A message is sent from the point-of-return to the Returns Management system indicating the item to be returned, if the customer has a receipt, and possibly other data. See "Point of Return to Returns Management—Initial Return Request".
3. Returns Management chooses which policy to initiate. A policy is comprised of one or more rules, and each policy has associated metadata that enables the service layer to choose the most appropriate policy for the current item in question. See "Determining Return Policies" in *Oracle Retail Returns Management User Guide*.
4. Returns Management gathers together relevant server-side historical information, such as entries related to the customer in the exception file.
5. The policy might require additional data from the point-of-return, such as a positive ID from the customer. In this case, a message is sent back to the point-of-return asking for the additional data. See "Returns Management to Point of Return—Initial Return Response: Need Positive ID".

6. The policy decides if the item is returnable or not. Returns Management informs the point-of-return of its decision, and provides a tender recommendation.

   The point-of-return ultimately decides to accept the return or not. For example, Returns Management might say that an item is non-returnable, but a local manager might override that decision. A local manager can also ignore a tender recommendation.

7. The point-of-return informs Returns Management of its decision. See "Point of Return to Returns Management—Return Result from Second Response".

8. Returns Management uses information from the point-of-return to update its historical records such as the exception file. The exception file acts as a constantly evolving knowledge base that can help the analytic engine decide which customers, items, cashiers, or stores are at higher risk for return fraud.

Conceptual Data Flow

To understand how the various modules relate to each other at run time, imagine the flow of data through the system.

- The four main processes (return request, return results, policy administration, and auditing) operate on an intersecting set of data.
- Return requests are sent to Returns Management and cause return tickets to be created and rules to be read.
- Rule initiation creates entries in the audit log.
- Return responses are sent back to the point-of-return.
- Return results update existing return tickets, and create entries in the exception file.

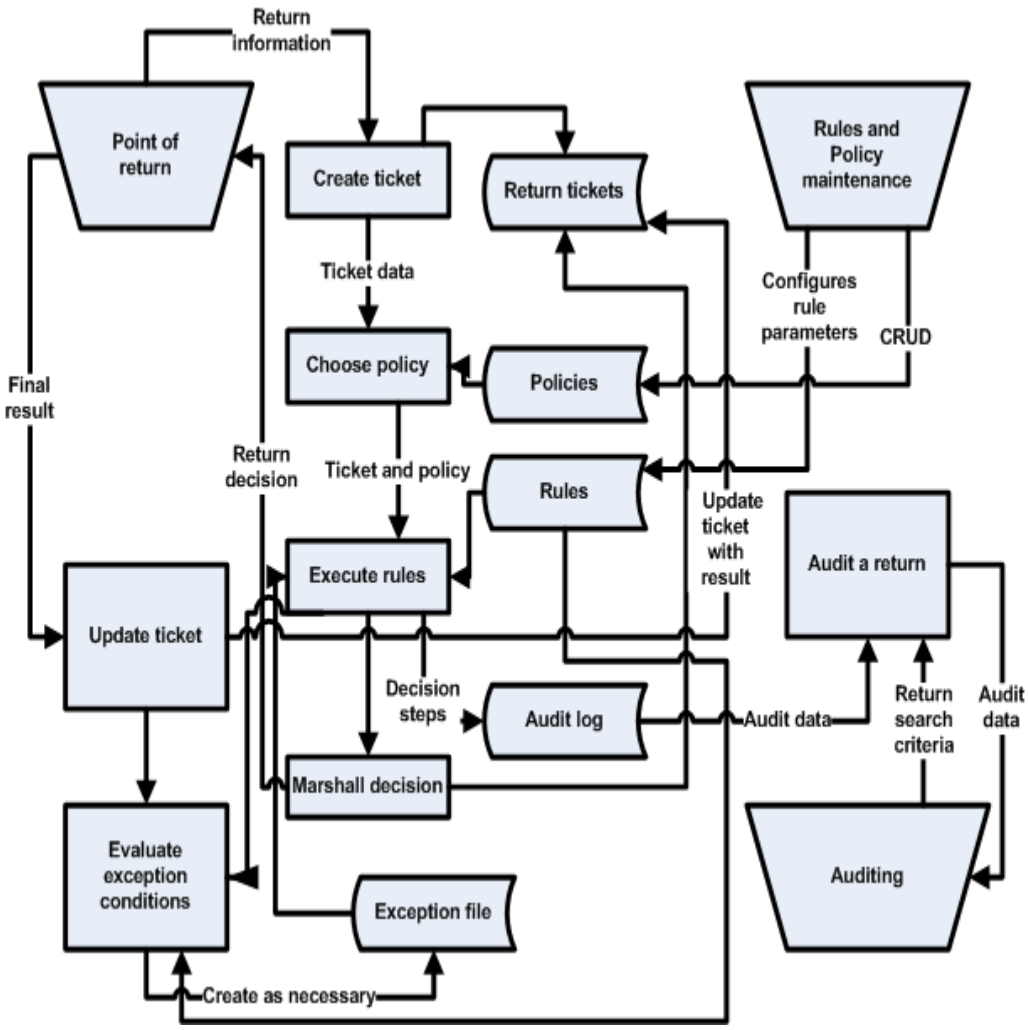
Policy administration enables the creation and maintenance of policies. See "Determining Return Policies" in the *Oracle Retail Returns Management User Guide*.

Auditing applications read the audit entries created during the evaluation phase.

Figure 4–2 lists the four main processes and the flow of the data that they create and consume.

Note: The Rules and Policy Maintenance interface is shown as directly updating the policy rules using Create, Read, Update, Delete (CRUD).

Figure 4–2 Oracle Retail Returns Management Conceptual Data Flow



Functional Assumptions

- Though Returns Management needs to be informed of returns performed, it is not a requirement that the point-of-return itself informs Returns Management. This means that this information can be conveyed by a separate process, such as a scheduled transaction parsing routine. This also means that the point-of-return does not need to have direct access to the process final result API of Returns Management.
- The historical data read by Returns Management at the beginning of a return request is the data that is updated by the return information delivered after a successful return.
- The historical data recorded by Returns Management is not the same as purely transactional data, for example, POSLog. The historical data is data that reflects inferred customer behavior, such as too many returns over a specified amount of time.
- The retail transaction data is read by the point-of-return, not by Returns Management.

Functional Overviews

The following are different functional overviews.

Return Tickets Functional Overview

Return tickets enable an operator to inquire about the particulars of a specific return approval or denial. A return ticket is any attempt by a customer to return one or more items, from one or more originating transactions or from no identifiable transaction. The return ticket carries a unique identifier that can consist of the store number and workstation ID from which the return attempt occurs, an eight-digit date in *MMDD-YYYY* format, and sequence number. The operator can search for a return ticket by the unique identifier or other information, such as cashier, customer, or item information.

The operator can be a loss prevention operator researching potentially fraudulent return activity, or a customer service person researching why a particular customer's return was denied.

Exception Files Functional Overview

The Returns Management exception file is created and maintained by Returns Management for use in detecting and preventing fraud at the point-of-return. The exception file acts as a constantly evolving knowledge base that can help the authorization engine decide which customers or cashiers are at higher risk for return fraud.

Exceptions are instances of a behavior that a retailer has selected to track for a customer or cashier. The exception file holds an exception counter for a customer; the exception counter is incremented based on suspicious return activity. If a return activity is selected for inclusion in the exception counter, the system increments the exception count for each suspicious shopping activity. Likewise, return activities can be configured for cashiers.

When an exception occurs, a record is written to the exception file and the activity is available for research on that customer or cashier using the exception inquiry search and display screens. All exceptions are based on return ticket data.

Exception counts are based on real-time refund attempt activities occurring at the point-of-return, using the return result message that is sent by the point-of-return to Returns Management at the conclusion of a transaction with an attempted refund. Return activities include activities that increment counters such as a return transaction by the customer without a receipt and with no retrieval of the original transaction, five same-day returns as purchases within the last three days, and three returns today. In turn, normal activity levels might be exceeded and counting generated based on those counters.

Messages and Responses Functional Overview

The message and response component of Returns Management includes the messages sent from the point-of-return that might trigger action in Returns Management and an appropriate response message. Returns Management communicates with brick and mortar, e-commerce, and call center point-of-return environments using a messaging interface to receive return authorization requests, use retailer-defined return policies to determine authorization or denial of items and valid return tenders, and respond with the applicable approval or denial code.

Policies and Rules Functional Overview

A return policy consists of multiple rules that ask a question about an attempted return. The retailer sets the order in which the rules are evaluated upon a return. The retailer determines the action to take based on the answer to the question. The action taken based on the answer to the question is:

- Continue
- Continue At Rule Number
- Stop Processing

Analytic Engine Functional Overview

The following is an overview of the analytic engine.

Configuration

A return policy consists of multiple rules that ask a question about an attempted return, such as some of the following:

- Does the customer have a receipt?
- Is the item serialized?
- Does the serial number on the item being returned match the serial number of the item as originally purchased?
- What is the customer's cumulative exception count?
- What is the condition of the item?

The retailer sets the order in which the rules are evaluated upon a return.

The retailer determines through the front end the action to take based on the answers to the questions.

The answer can be **Yes** or **No** (Boolean), a certain numeric or currency number (Range), or one possible response from a valid list of responses (Discrete), based on the type of question being asked. For example, "Does the customer have a receipt?" has a **Yes** or **No** response. "What is the customer's cumulative exception count?" has a numeric response that would fall within a range configured by the retailer. "What is the condition of the item?" maps to a response chosen by the point-of-return operator, such as one of the following:

- Excellent
- Good
- Fair
- Poor
- Open Box
- Damaged
- Used

The analytic engine uses one of these items to decide returnability.

The action taken based on the answer to the question is one of the following:

- Continue—Check the next rule within the policy.
- Continue At Rule Number—Check a particular rule within the policy and then continue.
- Stop Processing—Do not continue checking rules. Processing complete.

Response Codes

The retailer can set a configurable response code to be returned with every action. Response codes consist of a required positive numeric code, response type, response priority within that type, short description, and an optional long description that can be used for scripting customer service responses to customer inquiries. The response type for each response code is selected from the following, which are listed in priority order:

- Denial.
- Manager Overridden Denial—The engine has denied the item but the denial can be overridden at the point-of-return by a properly authorized user.
- Contingent Authorization—The engine has approved the item contingent upon capture of an override at the point-of-return by a properly authorized user.
- Authorization.

Response codes are prioritized within response types. No two response codes of the same response type can have the same priority. As the analytic engine evaluates policy rules, the system holds the highest priority response code within that response type as the response, until a rule resulting in a higher response type, with a higher priority, supersedes it, thus the retailer can control whether the most favorable or least favorable response is returned to the point-of-return.

Tender Determination

The retailer also determines the tenders that are enabled for a return. When the response is Continue or Continue At Rule Number, the tenders set for the rule carry forward until they are superseded by the response to a following rule. If there is no following rule that must be evaluated, then the tenders collected as a response to that rule are the available tenders that are returned to the point-of-return in the response message.

Collection of Customer Demographics

An indicator can be set on a policy rule response that indicates positive ID is required in order to check this rule, and the policy cannot be evaluated unless the positive ID is obtained and the exception file checked. In this case, an additional call to Returns Management is made, for another evaluation once customer positive ID is obtained.

Determination of the Policy for Use on a Return Attempt

The collection of rules (policy) is assigned to a combination of location (node of the store reporting hierarchy, ad hoc store groups, or individual stores) and items, which can be designated by item or merchandise hierarchy. When a return is attempted at a point-of-return, the system determines the appropriate policy to apply based on the item being returned and the store where the return is being performed. The item designation supersedes the store designation in the case where two policies might otherwise be equivalent.

Two default policies must be defined for the analytic engine to use:

- Receipted items
- Non-receipted items

Exception policies can then be set to cover specific items, such as serialized items that include warranties, or articles of clothing that cannot be returned under any condition. When the system does not find a policy applying specifically to the line item being returned, the system falls back to the appropriate default receipted or non-receipted policy to evaluate returnability.

When the returnability has been determined based on the appropriate policy, the system checks for any other items that the customer is attempting to return at that time. When the responses have been determined for all items in the attempted return, the system sends the return response message with the evaluation results for the attempted return. The response for an attempted line item return includes a response code and description that are determined by the retailer.

The point-of-return can then use the response information to control flow to complete the return, such as prompting for a manager override, presenting the enabled tenders, or displaying information for why a return is not allowed.

Customer Service Overrides

Customer service overrides are granted to a customer using the Customer Exception Details screen. The presence of a customer service override for a particular positive ID is checked at the end of return engine evaluation if any line item evaluates to a Manager Overridden Denial, or Denial. Customer service overrides are associated and used with a return ticket. If the return ticket is subsequently voided, the customer service override is considered unused and might be used with a subsequent return authorization. If more than one customer service override exists, the system applies them to the return in order from oldest to newest, by date.

Customer service overrides can consist of more than one allowed return within an override. The Max Customer Service Overrides parameter limits the number of allowed returns within the override and the total number of overrides granted to a customer.

5

Store Database

Point-of-Service Store Database

This chapter describes the database used with Point-of-Service and how to interface with it, including:

- Updating tables
- Rebuilding the database
- Creating new tables
- Updating flat file configurations

The chapter includes an example of writing code to store new data in the database.

ARTS Compliance

The Point-of-Service system uses an Association of Retail Technology Standards (ARTS)-compliant database to store transactions and settings. The ARTS standard (see http://www.nrf-arts.org/) is a key element in maintaining compatibility with other hardware and software systems.

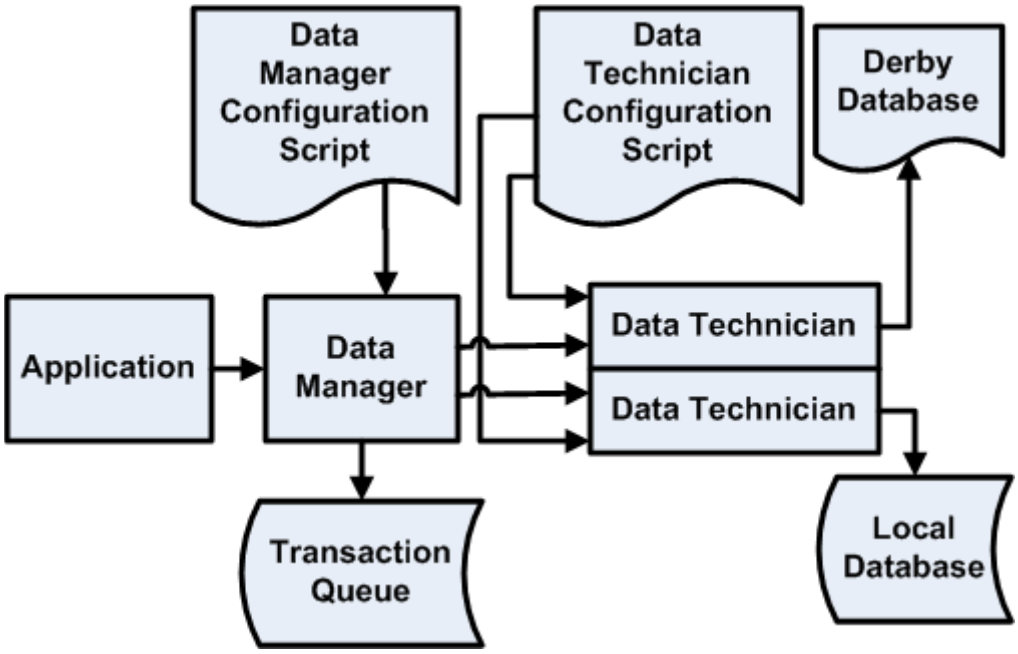
Although the Point-of-Service system complies with the ARTS guidelines, it does not implement the entire standard, and contains some tables which are not specified by ARTS. For example, ARTS tables for store equipment and recipe are not included, while tables for tender types and reporting have been added.

The ARTSDatabaseIfc.java file defines the mapping of ARTS names to constants in application code.

Understanding Data Managers and Technicians

Figure 5–1 shows how Data Managers and Data Technicians handle communication with the database in the Point-of-Service application.

Figure 5–1 Data Managers and Data Technicians



The Point-of-Service system uses the following components to write to the database:

- The Data Manager's primary responsibilities are to provide an API to the application code and to contact the Data Technician and pass it data store requests.

  There is one Data Manager per client. The Data Manager manages connections to multiple Data Technicians, for example, there is a Data Technician residing on the client that retrieves data from the offline (Derby) database, and there is also a Data Technician residing on the store server that manages access to the store database. The Data Manager on the client is configured to determine which Data Technician provides which data service.
- The Data Manager Configuration Script is an XML file that specifies the properties of the Data Manager.
- The Data Technician handles the database connection. Configure the Data Technician with an XML script. The Data Transaction class is the valet from the manager to the technician. The Data Transaction class has the add, find, and update methods to the database. Typically, there is one Data Technician that communicates with the local database and one that communicates with offline database.

  Note: Most managers create valets when they need talk to technicians. Data Manager works a little differently: the Data Transaction class calls the Data Manager and passes itself as a valet. The valet finds the data operation class, then the valet knows which technician it is associated with and calls its execute method.

- The Data Technician configuration script is an XML file that specifies the properties of the Data Technician.
- The Transaction Queue collects data transactions and guarantees delivery.

- Offline Database is the Derby database that is used when the register is offline.
- The Local Database is the store database.

How Data Transactions Work

This section gives an overview of how Oracle Retail Platform, Data Manager, and Data Technician components work together to store data in the database.

> **Note:** The notation TXN refers to a data transaction, which can be any guaranteed transmission of data, not necessarily a sales transaction in the retail sense.

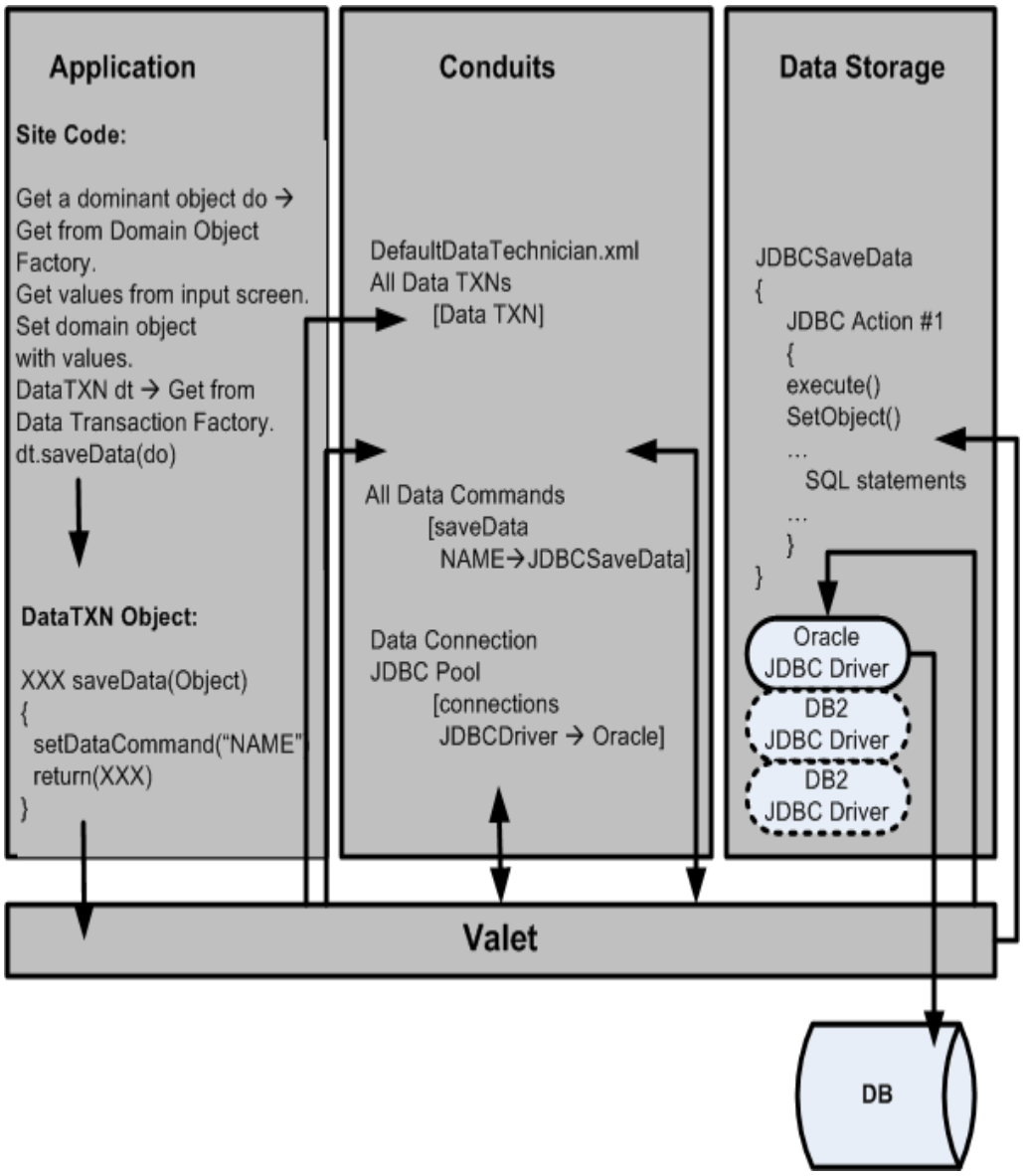
Oracle Retail Platform is responsible for configuring the system so that the Data Manager, Data Technician, configuration scripts, and conduit scripts work together to provide the mechanism to update, store, and retrieve data from a database.

1. The client conduit script defines the name and package for the Data Manager and Data Manager configuration script, POSDataManager.xml.
2. The server conduit script defines the name and package for the Data Technician and Data Technician configuration script, DefaultDataTechnician.xml.
3. At runtime, the tour code requests a data transaction object from the Data Transaction Factory.
4. The Data Transaction Factory verifies that the transaction is defined in POSDataManager.xml and the transaction object is returned to the tour code.
5. The tour code calls a method on the transaction object that creates a vector of data actions. A data action corresponds to a set of SQL commands that are executed as a unit. (Data actions are reused by different transactions.)
6. The method in the transaction object gets a handle to the Data Manager and calls execute(), sending itself as a parameter. This instructs the Data Manager to send the Transaction object (a valet) across the network to the Data Technician.

   > **Note:** Most Manager/Technician pairs work differently. The standard pattern is for the tour code to get a handle to the Manager, then call a method on the manager that will create the valet object and send it to the technician. For the Data Manager/Technician pair, the transaction object (the valet class), gets the handle to the Data Manager. The tour code is only responsible for getting a transaction object from the factory and calling the appropriate method.

7. On the server side, the Data Technician configuration script, DefaultDataTechnician.xml, lists all available transactions. It also defines an operation class for each data action. Each data action is then processed by the appropriate data operation class.

Figure 5–2 Updating the Database: Simplified Runtime View



Note:

- The DataTechician can be configured to write an error file for each failure that is not a connection error.
- The DataManager can be configured to delete the head of the queue when the failure is not a connection error.

See the DefaultDataTechnician.xml file. This file contains the following element at the end of the file:

```
<EXCEPTIONHANDLER class="SQLExceptionHandler"
     package="oracle.retail.stores.domain.manager.data"/>
```

See the DataManager.xml file. This file contains the following element at the end of the file:

```
    <QUEUE name="TransactionQueue"
              encryptBuffer="true"
            class="DataTransactionFileQueue"
          package="oracle.retail.stores.foundation.manager.data">
           <EXCEPTIONHANDLER
class="TransactionQueueSQLExceptionHandler"
package="oracle.retail.stores.domain.manager.data"/>
    </QUEUE>
```

The exception handling classes are implemented as plug points.

Creating or Updating Database Tables

Use this procedure when creating a new database table or updating an existing one. Refer to the ARTS standards when designing tables.

Note: When you add or change a table, you need to rebuild the database for your local copy of Point-of-Service before you can test your changes. See Step 6.

1. Edit the appropriate database script, or write a new one.

   Database scripts can be found in the source directory <*source_directory*>\modules\common\deploy\server\common\db\sql.

   Start a new file (or edit the appropriate existing file) in the db/sql source directory file to store SQL commands for creating the new table. Example 5–1 shows the SQL commands for creating the table that stores the credit card data.

Example 5–1 CreateTableCreditDebitCardTenderLineItem.sql

```
DROP TABLE TR_LTM_CRDB_CRD_TN;

CREATE TABLE TR_LTM_CRDB_CRD_TN
(
    ID_STR_RT           char(5) NOT NULL,
    ID_WS               char(3) NOT NULL,
    DC_DY_BSN           char(10) NOT NULL,
    AI_TRN              integer NOT NULL,
    AI_LN_ITM           smallint NOT NULL,
    TY_TND              varchar(20),
```

```
    ID_ISSR_TND_MD      varchar(20),
    TY_CRD              VARCHAR(40),
    ...additional column lines omitted here...
);

ALTER TABLE TR_LTM_CRDB_CRD_TN ADD PRIMARY KEY (ID_STR_RT, ID_WS, DC_DY_BSN, AI_
TRN,
AI_LN_ITM);

COMMENT ON TABLE TR_LTM_CRDB_CRD_TN IS 'Credit/Debit Card Tender Line Item';

COMMENT ON COLUMN TR_LTM_CRDB_CRD_TN.ID_STR_RT IS           'Retail Store ID';
COMMENT ON COLUMN TR_LTM_CRDB_CRD_TN.ID_WS IS               'Workstation ID';
COMMENT ON COLUMN TR_LTM_CRDB_CRD_TN.DC_DY_BSN IS           'Business Day Date';
COMMENT ON COLUMN TR_LTM_CRDB_CRD_TN.AI_TRN IS              'Transaction Sequence
Number';
COMMENT ON COLUMN TR_LTM_CRDB_CRD_TN.AI_LN_ITM IS           'Retail Transaction
Line Item
Sequence Number';
COMMENT ON COLUMN TR_LTM_CRDB_CRD_TN.ID_ISSR_TND_MD IS      'Tender Media Issuer
ID';
COMMENT ON COLUMN TR_LTM_CRDB_CRD_TN.TY_TND IS TenderTypeCode';
COMMENT ON COLUMN TR_LTM_CRDB_CRD_TN.TY_CRD IS              'Card Type';
...additional comment lines omitted...
```

2. Create or edit the insert files (also in the db/sql source directory) for inserting initial data into the new database table.

   This step is used only to insert data into the database table for purposes of initially logging on, testing, and so on.

3. Make updates to foreign keys in CreateForeignKeys.sql.

4. If you are creating a new table, add a string constant to the <*source_directory*>\modules\common\src\oracle\retail\stores\persistence\utility\ARTSDatabaseIfc.java file. Use a string constant with a meaningful name to store the official ARTS name of the database table.

   Example 5–2 shows two examples of meaningful String constants found in ARTSDatabaseIfc.java.

Example 5–2 String Constant in ARTSDatabaseIfc.java

```
public static final String TABLE_TENDER_LINE_ITEM = "tr_ltm_tnd";
public static final String TABLE_CREDIT_DEBIT_CARD_TENDER_LINE_ITEM = "tr_ltm_
crdb_crd_tn";
```

5. Check foreign key constraints.

   For performance reasons, the database build scripts do not turn on foreign key constraints until late. If you make inserts which break foreign key constraints, you will not be notified. To check this, test all inserts with foreign key constraints in place, by editing the appropriate database build script.

6. Open a command prompt in the Point-of-Service installer directory and use the following command-lines:

   - To reset the store database: `install.cmd ant install-database`
   - To reset the scratchpad database: `install.cmd ant install-scratchpad`

- To reset both: `install.cmd ant install-database install-scratchpad`

The install-database command uses the settings in the ant.install.properties file, so the dataset specified by the input.install.database property is loaded. The values can be:

- no - no action taken
- schema – only install the schema, no data
- minimum – schema and minimum required data
- sample – schema, minimum, and sample data

To reset the scratchpad database, the ant.install.properties file needs to have the scratchpad database information as well as input.install.scratchpad.database set to true.

7. After you verify that the table builds successfully and the code referencing the table works, check your updates into source control.

Example of Saving Data: Storing Till Information

This section describes how to save data to the database, using till close information as an example.

Research Table Requirements and Standards

To plan your database code, refer to functional requirements documents to determine what data must be stored.

Next, review the ARTS database standards for tables and columns. Determine whether you need to create a new table. If you need to create a table defined by ARTS but not currently used in the Store database, follow the ARTS standard. For instructions on creating a new table, see "Creating or Updating Database Tables".

For the till transaction, there are several tables that need to be addressed: the tender line item table and the credit/debit card transaction table.

Table 5–1 lists database tables used in a credit card tender option.

Table 5–1 Database Tables Used in Credit Card Tender Option

ARTS Table Name	Description
TR_TL_OPN_CL	till open close transaction table
TR_CTL	control transaction table
LE_HST_STR_SF_TND	Store Safe Tender History table
AS_TL	the Till table
LE_HST_TL	till history
LE_HST_WS	workstation history
LE_HST_STR	store history

Saving Data from Site Code

To save data to the database from a site:

1. Create and populate the domain object to be saved.
2. Save the data to the cargo's transaction.

For the Till Close option, the TillCloseCargo contains the tillID of the till to close.

In Example 5–3, from <*source_directory*>\applications\pos\src\oracle\retail\stores\pos\services\dailyoperations\till\tillclose\UpdateStatusSite.java, Till is a domain object that stores the till data such as the expected amount and entered amount, and so on. In the following code, the till object is retrieved from the register object (stored in cargo) based on the till id updated and added to the TillOpenCloseTransaction line item.

Example 5–3 UpdateStatusSite.java: Transaction Object

```
public void arrive(BusIfc bus)
{
TillCloseCargo cargo = (TillCloseCargo) bus.getCargo();

        // Local references to register and till.
        RegisterIfc register = cargo.getRegister();
        TillIfc till = register.getTillByID(cargo.getTillID());

// create close till transaction
        TillOpenCloseTransactionIfc transaction =
            DomainGateway.getFactory().getTillOpenCloseTransactionInstance();
//save current register accountability in the register
        till.setRegisterAccountability(register.getAccountability());

//add the till to the transaction
transaction.setTill(till);
  ...
  // Add the credit line item to the transaction
  trans.addTender(credit);
  ...
}
```

3. Call a method to save the transaction object.

   After the till object is added to the TillOpenCloseTransactionIfc transaction, the collected data is saved to the database. In Example 5–4, the <*source_directory*>\applications\pos\src\oracle\retail\stores\pos\services\dailyoperations\till\tillclose\UpdateStatusSite.java file uses the Utility Manager to call the saveTransaction() method.

Example 5–4 SaveRetailTransactionAisle.java: Save Transaction

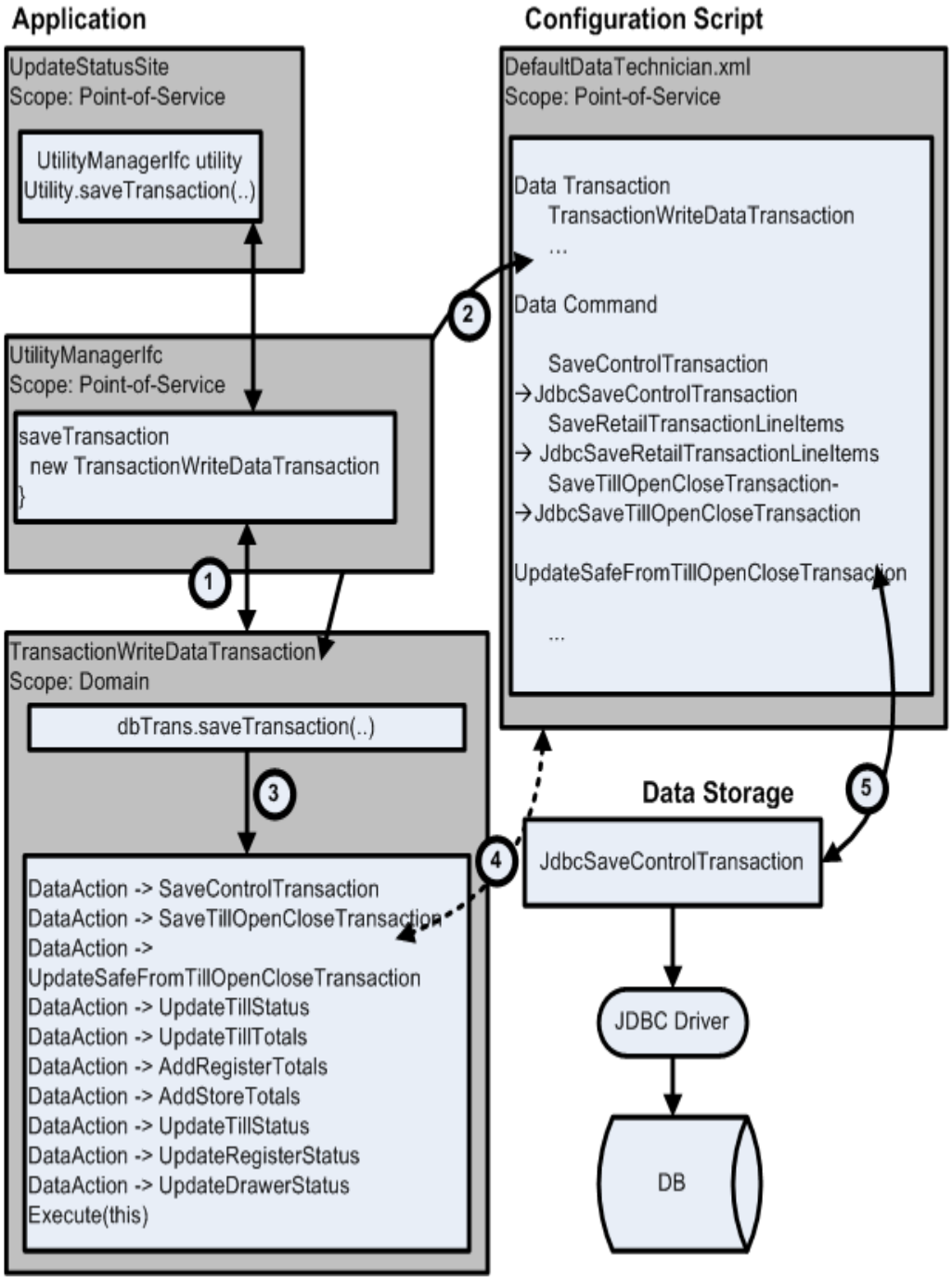
```
  ...
  UtilityManagerIfc utility =(UtilityManagerIfc)
bus.getManager(UtilityManagerIfc.TYPE);
  ...

// save the till close transaction
            utility.saveTransaction(transaction);
  ...
```

Locate Data Operation

The Data Manager and Data Technician work together to provide access to the database from the application. The developer rarely modifies these. Typically, the site code and the JDBC code are updated. To identify which JDBC class should be used, trace through the site code until the DataAction sets the operation name.

Figure 5–3 Diagram: Saving a Transaction



The following descriptions explain the labels in the figure.

1. UpdateStatusSite uses the Utility Manager to call the saveTransaction() method as shown in Example 5–4. The utility.saveTransaction() method uses the data transaction class TransactionWriteDataTransaction to save the till transaction.

   The following code samples show details for the previous figure.

Example 5–5 UtilityManager.java: Save Data Transaction

```
TransactionWriteDataTransaction dbTrans = new
TransactionWriteDataTransaction(tranName);
dbTrans.saveTransaction(trans, totals, till, register);
```

Example 5–6 TransactionWriteDataTransaction.java: Save Transaction

```
public void saveTransaction(TransactionIfc transaction,
                            FinancialTotalsIfc   totals,
                            TillIfc              till,
                            RegisterIfc          register)
                            throws DataException
{
  ...
  int transactionType = transaction.getTransactionType();
  ...
  switch(transactionType)
  {                         // begin add actions based on type
...
case TransactionIfc.TYPE_OPEN_TILL:
                    addSaveTillOpenTransactionActions(transaction);
                    break;
               case TransactionIfc.TYPE_CLOSE_TILL:
                    addSaveTillCloseTransactionActions(transaction);
                    break;
    ...
}
```

2. The <*source_directory*>\applications\pos\deploy\server\config\technician\DefaultDataTechnician.xml file is the configuration file for the Data Technician and is used to configure the links between the application and the JDBC class that performs the work. All Data Transaction classes must be defined in this file, including TransactionWriteDataTransaction.

Example 5–7 DefaultDataTechnician.xml: Define Data Transaction Class

```
<DATATECHNICIAN
  package="oracle.retail.stores.domain.arts">
  ...
  <TRANSACTION name="TransactionWriteDataTransaction" command="jdbccommand"/>
  ...
```

3. The TransactionWriteDataTransaction class instantiates the DataAction object and sets the data operation name to UpdateTillStatus and so on. Other data actions occurred before these till data actions. Data Actions are added in the specific order in which they should occur.

Example 5–8 TransactionWriteDataTransaction: DataAction

```
protected void addSaveTillCloseTransactionActions(TransactionIfc transaction)
// save the control transaction
        DataActionIfc dataAction = createDataAction(artsTransaction,
                                                    "SaveControlTransaction");
        actionVector.addElement(dataAction);

        // this ensures that the change is backward compatible, because
        // only if till open-close transaction is used will the new data
operations
        // be executed
```

```
        if (transaction instanceof TillOpenCloseTransactionIfc)
        {
            TillOpenCloseTransactionIfc tocTransaction =
              (TillOpenCloseTransactionIfc) transaction;
            // save the till open/close transaction
            dataAction = createDataAction(transaction,
                                          "SaveTillOpenCloseTransaction");
            actionVector.addElement(dataAction);
            // build ARTS till for other operations
            TillIfc till =
              ((TillOpenCloseTransactionIfc) transaction).getTill();
            RegisterIfc register =
              ((TillOpenCloseTransactionIfc) transaction).getRegister();

            if (till.getStatus() == AbstractFinancialEntityIfc.STATUS_RECONCILED)
            {
                // update the safe as needed
                dataAction = createDataAction(transaction,

"UpdateSafeFromTillOpenCloseTransaction");
                actionVector.addElement(dataAction);

                // Get deep copies of the till and register so they can be loaded
                // with the till-close totals
                TillIfc         aTill = (TillIfc) till.clone();
                RegisterIfc aRegister = (RegisterIfc) register.clone();

                // Combine the till and float totals objects
                FinancialTotalsIfc totals =
                  DomainGateway.getFactory().getFinancialTotalsInstance();
                totals.addEndingFloatCount(tocTransaction.getEndingFloatCount());
                totals.getCombinedCount().setEntered
                  (tocTransaction.getEndingCombinedEnteredCount());

                // Set the counted totals on the till and register.
                aTill.setTotals(totals);
                aRegister.setTotals(totals);
                ARTSTill artsTill = new ARTSTill(aTill, aRegister);

                // creates or updates the till as needed
                dataAction = createDataAction(artsTill,
                                              "UpdateTillStatus");
                actionVector.addElement(dataAction);

                // creates or updates the till totals as needed
                dataAction = createDataAction(artsTill,
                                              "UpdateTillTotals");
                actionVector.addElement(dataAction);
                // add to register totals
                dataAction = createDataAction(aRegister,
                                              "AddRegisterTotals");
                actionVector.addElement(dataAction);
                // add to store totals
                ARTSStore aStore = new
ARTSStore(register.getWorkstation().getStore(),
                                                register.getBusinessDate());
                aStore.setFinancialTotals(aRegister.getTotals());
                dataAction = createDataAction(aStore,
                                              "AddStoreTotals");
                actionVector.addElement(dataAction);
```

```
            }
            else
            {
                ARTSTill artsTill = new ARTSTill(till, register);

                // creates or updates the till as needed
                dataAction = createDataAction(artsTill,
                                              "UpdateTillStatus");
                actionVector.addElement(dataAction);
            }

            // update the register and drawer
            dataAction = createDataAction(register,
                                          "UpdateRegisterStatus");
            actionVector.addElement(dataAction);
            // update the drawer
            dataAction = createDataAction(register,
                                          "UpdateDrawerStatus");
            actionVector.addElement(dataAction);
        }
```

Example 5–9 UpdateTillStatus: Set Data Operation Name

```
protected static final String OPERATION_NAME = "UpdateTillStatus";
```

4. The DefaultDataTechnician uses the data command to list several data operation names. The data operation name UpdateTillStatus points to the name of the JDBC class, which is JdbcUpdateTillStatus.

Example 5–10 DefaultDataTechnician.xml: Define Data Operation Class

```
<DATATECHNICIAN
  package="oracle.retail.stores.domain.arts">
  ...
  <TRANSACTION name="TransactionWriteDataTransaction" command="jdbccommand"/>
  ...
  <COMMAND name="jdbccommand"
           class="DataCommand"
           package="oracle.retail.stores.foundation.manager.data"

  <COMMENT>
    This command contains all operations supported on a JDBC
    database connection.
  </COMMENT>
  <POOLREF pool="jdbcpool"/>
  ...
  <OPERATION class="JdbcUpdateTillStatus"
    package="oracle.retail.stores.domain.arts"
    name="UpdateTillStatus">
    <COMMENT>
      This operation updates the till status in the database.
    </COMMENT>
  </OPERATION>
...
</DATATECHNICIAN>
```

5. The JdbcUpdateTillStatus class is used to update the till status to the database table. See the next section.

Modify Data Operation

Use this procedure to modify the data operation class to access the database.

1. Add a save method to the data operation class.
2. Write an implementation for methods written for the data operation class.

   Second, the credit data must be saved to the new database table using SQL factory methods.

Example 5–11 JdbcUpdateTillStatus.java: SQL Factory Methods

```
 public boolean updateTill(JdbcDataConnection dataConnection,
                           TillIfc till,
                           RegisterIfc register)
                           throws DataException
    {
        boolean returnCode = false;
        SQLUpdateStatement sql = new SQLUpdateStatement();
        isUpdateStatement = true;

        /*
         * Define the table
         */
        sql.setTable(TABLE_TILL);

        /*
         * Add columns and their values
         */
        sql.addColumn(FIELD_TILL_SIGNON_OPERATOR,
makeSafeString(till.getSignOnOperator().getEmployeeID()));
        if (till.getSignOffOperator() != null)
        {
            sql.addColumn(FIELD_TILL_SIGNOFF_OPERATOR,
makeSafeString(till.getSignOffOperator().getEmployeeID()));
        }
        sql.addColumn(FIELD_TILL_STATUS_CODE, getStatusCode(till));
        sql.addColumn(FIELD_TILL_STATUS_DATE_TIME_STAMP,
dateToSQLTimestampString(new Date()));
        sql.addColumn(FIELD_WORKSTATION_ID, getWorkstationID(register));
        sql.addColumn(FIELD_TILL_START_DATE_TIMESTAMP, getStartTimestamp(till));
        sql.addColumn(FIELD_BUSINESS_DAY_DATE,
getBusinessDay(till.getBusinessDate()));
        sql.addColumn(FIELD_WORKSTATION_ACCOUNTABILITY, "'" +
till.getRegisterAccountability() + "'");
        sql.addColumn(FIELD_TILL_TYPE, "'" + till.getTillType() + "'");
        /*
         * Add Qualifier(s)
         */
        sql.addQualifier(FIELD_RETAIL_STORE_ID + " = " + getStoreID(register));
        sql.addQualifier(FIELD_TENDER_REPOSITORY_ID + " = " + getTillID(till));

        try
        {
            dataConnection.execute(sql.getSQLString());
        }
        catch (SQLException se)
        {
            logger.error( "" + se + "");
            throw new DataException(DataException.SQL_ERROR, "Update Till", se);
        }
```

```
        if (0 < dataConnection.getUpdateCount())
        {
            returnCode = true;
        }

        return(returnCode);
    }
```

Test Code

To test the new code:

1. Run Point-of-Service.
2. Select the path to the screen.
3. Enter the data.
4. Complete the till close.

Verify Data

To verify that the correct data exists in the database table, use a database access program to view the table that should contain the new information. Verify that the data in the database table matches the data entered. The following example shows a sample SQL statement you can use to retrieve the data.

```
select * from AS_TL;
```

Central Office and Back Office Store Database

Point-of-Service uses an ARTS-compliant database. Data is stored and retrieved by entity beans in a bean-managed persistence pattern, so the system makes database calls from the entity bean code.

A single entity bean exists for each database table, and handles reads and writes for that table. Each entity bean contains the necessary methods to create, load, store, and remove its object type.

The Central Office application writes data to the enterprise database, which serves as a repository for information about transactions across the whole enterprise.

The Back Office application writes data to the Store database, a repository for transaction information for a single store.

A DAO (data access object) provides an abstract interface to the underlying database tables. It accesses one or more tables that belong to the same logical unit to read and write information to the database.

Related Documentation

Table 5–2 lists related sources that provide specific information about the database for your use when developing code.

Table 5–2 Related Documentation

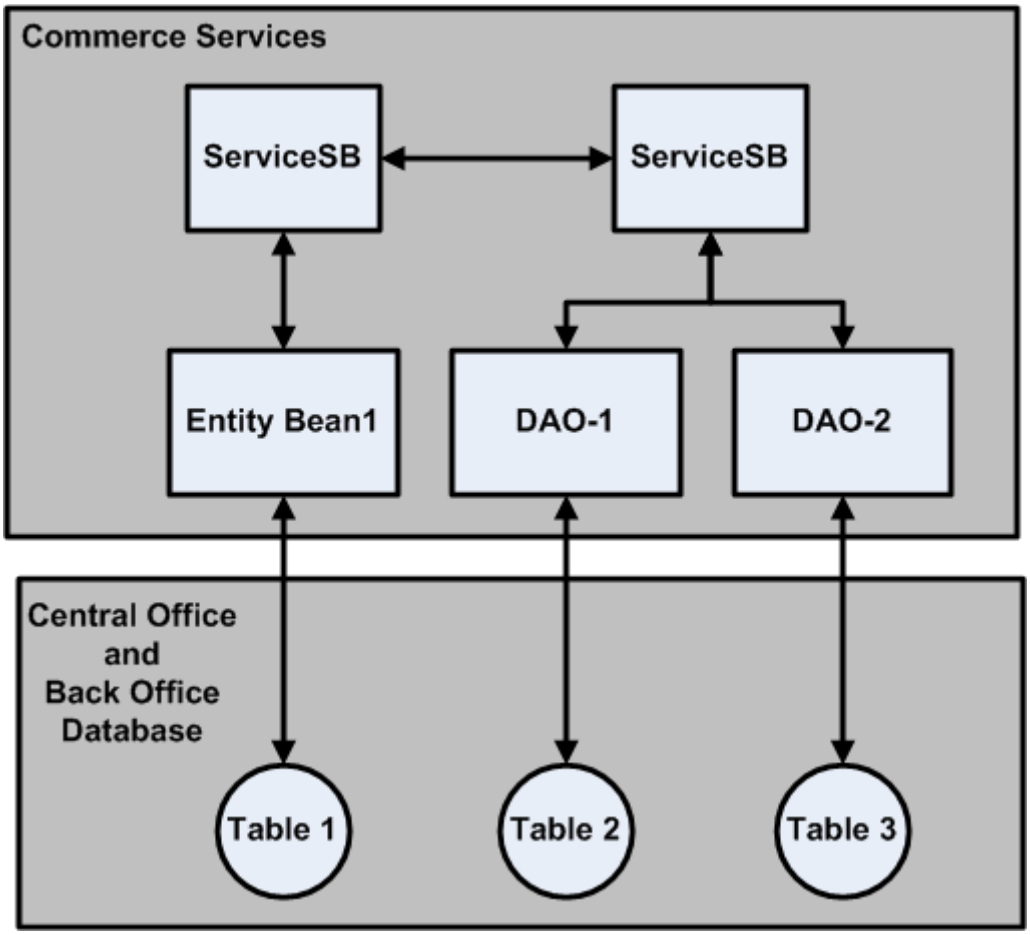
Source	Description
ARTS Database Standard	See http://www.nrf-arts.org/ for a description of the ARTS database standard.
Data Dictionary	Contains table and column definitions for the database used to store Point-of-Service data. See the docs zip file provided with your Point-of-Service documentation.
Database Diagrams	Diagrams which show the relationships between various tables in the database schema. See the docs zip file provided with your Point-of-Service documentation.

Database/System Interface

As described in Chapter 2, "Technical Architecture" a persistence layer of entity beans represents the database tables to the rest of the system. One bean represents each table.

Figure 5–4 illustrates these relationships.

Figure 5–4 Commerce Services, Entity Beans, and Database Tables



Each commerce service communicates with one or more entity beans or DAOs, and each entity bean communicates with one database table. A DAO typically communicates with one or more tables that belong to the same logical grouping.

Although there are exceptions, in general only one commerce service communicates with an entity bean; other services request the information from the relevant service rather than talking directly to the entity bean. For example, if the Customer Service needs information provided by the Item Bean, it makes a request to the Item Service.

ARTS Compliance

When new code is added or features are added, modified, or extended, database plans should be evaluated to ensure that new data items fit the ARTS schema. Complying with the standards increases the likelihood that extensions can migrate into the product codebase and improves code reuse and interoperability with other applications.

Note: Because the ARTS standard continues to evolve, older code may contain deviations from the standard or may be compliant only with an earlier version of the standard. Oracle Retail continues to evaluate ARTS compliance with each release of its software.

Bean-Managed Persistence in the Database

In general, the system uses standard J2EE bean-managed persistence techniques to persist data to the Oracle Retail database. Each of the entity beans that stores data requires JDBC code in standard ejbLoad, ejbStore, ejbCreate, and ejbRemove classes. However, there are some differences worth noting:

- All SQL references are handled as constant fields in an interface.
- Session and entity beans extend an EnterpriseBeanAdapter class. Special extensions for session and entity beans exist. These contain common code for logging and a reference to the Oracle Retail DBUtils class (which provides facilities for opening and closing data source connections, among other resources).

Example 5–12 ItemPriceDerivationBean.java: ejbStore Method

```
public void ejbStore() throws EJBException
    {
        ItemPriceDerivationPK key = (ItemPriceDerivationPK)
getEntityContext().getPrimaryKey();
        getLogger().debug("store");
        PreparedStatement ps = null;
        Connection conn = null;
        if (isModified())
        {
            getLogger().debug("isModified");
            try
            {
                conn = getDBUtils().getConnection();
                ps = conn.prepareStatement(ItemPriceDerivationSQLIfc.STORE_SQL);
                int n = 1;
                ps.setBigDecimal(n++,getReductionAmount().toBigDecimal());
                ps.setBigDecimal(n++,getDiscountPricePoint().toBigDecimal());
                getDBUtils().preparedStatementSetDate(ps, n++,
getRecordCreationTimestamp());
                ps.setBigDecimal(n++,getReductionPercent().toBigDecimal());
                getDBUtils().preparedStatementSetDate(ps, n++,
getRecordLastModifiedTimestamp());
                ps.setInt(n++, key.getPriceDerivationRuleID());
                ps.setString(n++, key.getStoreID());
```

```
                if (ps.executeUpdate() != 1)
                {
                    throw new EJBException("Error storing (" +
getEntityContext().getPrimaryKey() + ")");
                }
                setModified(false);
            }
            catch (SQLException ex)
            {
                getLogger().error(ex);
                throw new EJBException(ex);
            }
            catch (Exception ex)
            {
                getLogger().error(ex);
                throw new EJBException(ex);
            }
            finally
            {
                getDBUtils().close(conn, ps, null);
            }
        }
    }
```

DAO-Managed Persistence in the Database for Back Office

The system uses DAO persistence techniques to persist data to the database.

A DAO (data access object) provides an abstract interface to the underlying database tables. It accesses one or more tables that belong to the same logical unit to read and write information to database

The following is an example of a DAO:

Example 5–13 PluItemDAO

```
public PluItem getById(String storeId, String posItemId, String itemId, Locale
lcl) throws DataException
    {
        PluItem pluItem = null;

        Locale bestLocale = LocaleMap.getBestMatch(lcl);

        PreparedStatement ps = null;
        ResultSet rs = null;
        Connection conn = null;
        try
        {
            logger.debug("LOAD_PLU_ITEM_BY_ID_SQL " + LOAD_PLU_ITEM_BY_ID_SQL);

            conn = getDBConnectionManager().getConnection();
            ps = conn.prepareStatement(LOAD_PLU_ITEM_BY_ID_SQL);
            ps.setString(1, bestLocale.toString());
            ps.setString(2, bestLocale.toString());
            ps.setString(3, storeId);
            ps.setString(4, itemId);
            ps.setString(5, posItemId);

            rs = ps.executeQuery();

            if (rs.next())
```

```
            {
                pluItem = getPluItem(rs, true, bestLocale);
            }
        }
        catch (SQLException e)
        {
            logger.error(e);
            throw new DataException(DataException.SQL_ERROR, "failed to load item.
" + ", storeId = " + storeId
                    + ", posItemId = " + posItemId + ", itemId = " + itemId + ",
locale = " + bestLocale.toString(), e);
        }
        finally
        {
            getDBConnectionManager().close(conn, ps, rs);
        }
        return pluItem;
    }
```

6

Backend System Administration and Configuration

This chapter covers options for configuring Back Office, Central Office, Labels and Tags, Returns Management and Point-of-Service normally carried out by an administrator before the system goes into general use. It covers the following topics:

- Loading Oracle Retail Labels and Tags
- Parameters
- Running Back Office or Central Office
- Establishing a Store Hierarchy in Central Office or Returns Management
- Point-of-Service Devices
- Scheduling Post Processors in Back Office
- Data Management in Central Office
- Help Files in Point-of-Service
- Reason Codes in Point-of-Service
- Configuring Transaction ID Lengths
- Configuring RMI Timeout Intervals in Point-of-Service
- System Settings in Point-of-Service
- Configuring Logging in Point-of-Service

> **Note:** The *Oracle Retail POS Suite Security Guide* describes specific security features and implementation guidelines for the POS Suite products.

Loading Oracle Retail Labels and Tags

For loading Labels and Tags, Oracle Retail provides an SQL script, `ant init_labels`. This script is located in the `backofficeDBInstall.jar` file.

Parameters

The following information is about parameters in the Oracle Retail POS Suite.

Parameters in Back Office and Central Office

For more information about importing an initial set of parameters, see the *Oracle Retail Point-of-Service Installation Guide*.

For information on specific parameters, see the *Oracle Retail POS Suite Configuration Guide*.

Parameters in Point-of-Service

Parameters are used to control flow, set minimums and maximums for data, and allow flexibility without recompiling code. A user can modify parameter values from the UI without changing code. Parameter values can be modified by Point-of-Service, or the changes can be distributed by other Oracle Retail applications. For example, the maximum cash refund allowed and the credit card types accepted are parameters that can be defined by Point-of-Service. To configure parameters, you need to understand the parameter hierarchy, define the group that the parameter belongs to, and define the parameter and its properties.

Parameter Hierarchy

Parameters are defined in XML files that are organized in a hierarchy. Different XML files represent different levels in a retail setting at which parameters may be defined. Understanding the parameter hierarchy helps you define parameters at the appropriate level.

Table 6–1 lists the parameter directories, XML filenames, and file descriptions.

Table 6–1 Parameter Directories, Files, and Descriptions

Directory	Parameter-Related XML File	Description
application	application.xml	Default parameter information provided by the base product
corporate	corporate.xml	Company information
store	store.xml	Local store information
register	workstation.xml	Register-level information
user role	operator.xml	User-level information

Higher-level parameters by default are overridden by lower-level parameter settings. For example, store-level configuration parameters override application-level parameters. The FINAL element in a parameter definition signifies whether the parameter can be overridden. Example 6–1 is an excerpt from `<source_directory>\applications\pos\deploy\client\config\technician\PosParameterTechnican.xml`, showing the order of precedence from highest level to lowest level.

Example 6–1 Default Parameter Settings

```
<SELECTOR name="defaultParameters">
        <SOURCE categoryname="application" alternativename="application">
        <SOURCE categoryname="corporate" alternativename="corporate">
        <SOURCE categoryname="store" alternativename="store">
        <SOURCE categoryname="register" alternativename="workstation" >
        <SOURCE categoryname="userrole" alternativename="operator" >
    </SELECTOR
```

The `categoryname` specifies the directory name and the `alternativename` specifies the name of the XML file. All parameter subdirectories reside in `config\parameter`.

Parameter Group

Each parameter belongs to a group, which is a collection of related parameters. The groups are used when modifying parameters within the UI. The user selects the group first, then has the option to modify the related parameters that belong to that group. Examples of groups are Browser, Customer, Discount, and Employee.

Adding a parameter requires adding it to the proper group. The following excerpt from application.xml shows the Tender group and a parameter definition inside the group. The hidden attribute indicates whether or not the group is displayed in the UI.

Example 6–2 Definition of Tender Group

```
<GROUP name="Tender"
            hidden="N">
      <PARAMETER name="MaximumCashChange"
         ...
      <PARAMETER>
...
<GROUP>
```

Parameter Properties

Each parameter file contains parameter definitions organized by group. The following shows an example of two parameter definitions from `config/parameter/application/application.xml`.

Example 6–3 Parameter Definitions From application.xml

```
<PARAMETER name="CashAccepted"
         type="LIST"
         default="USD"
         final="N"
         hidden="N">
         <VALIDATOR class="EnumeratedListValidator"
            package="oracle.retail.stores.foundation.manager.parameter">
            <!-- Use ISO 3 letter currency code -->
            <PROPERTY propname="member" propvalue="None" />
            <PROPERTY propname="member" propvalue="USD" />
            <PROPERTY propname="member" propvalue="CAD" />
         </VALIDATOR>
         <VALUE value="USD"/>
         <VALUE value="CAD"/>

<PARAMETER name="MaximumCashChange"
         type="CURRENCY"
```

```
      final="N"
      hidden="N">
      <VALIDATOR class="FloatRangeValidator"
         package="oracle.retail.stores.foundation.manager.parameter">
         <PROPERTY propname="minimum" propvalue="0.00" />
         <PROPERTY propname="maximum" propvalue="99999.99" />
      </VALIDATOR>
      <VALUE value="25.00"/>
   </PARAMETER>
```

The FINAL attribute indicates whether the property definition is final, meaning it cannot be overridden by lower-level parameter file settings. The VALUE element is the current setting of the parameter. If multiple values are set, that means the value of the parameter is a list of values.

Table 6–2 lists the four types of VALIDATOR classes.

Table 6–2 Validator Types

Validator	Description
EnumeratedListValidator	Determines whether a value supplied is one of an allowable set of values
FloatRangeValidator	Ensures that the value lies within the specified minimum and maximum float range
IntegerRangeValidator	Ensures that the value of a parameter lies within the specified minimum and maximum integer range
StringLengthValidator	Ensures that the value's length lies within the specified min and max length

Running Back Office or Central Office

Do the following to run Back Office or Central Office:

1. Verify that the application is running in the application server environment.
2. Access the following URL from a browser, specifying the application server hostname and port number where indicated:

   `https://<app-server-hostname>:<port number>/<application>/`

   Where `<application>` is `backoffice` or `centraloffice`.

   `<port number>` is typically 9080 for IBM WebSphere Application Server.

   `<port number>` is also 7002 for Oracle stack, and 9443 for IBM stack.

Running Returns Management

Do the following to run Returns Management:

1. Verify that the application is running in the application server environment.
2. Access the following URL from a browser, specifying the application server hostname and port number where indicated:

   `https://<app-server-hostname>:<app-server-port number>/returnsmanagement/`

Establishing a Store Hierarchy in Central Office or Returns Management

Use the Data Import module to import store hierarchy information.

The store hierarchy defines where stores fit in the retailer's enterprise. The store hierarchy is defined in an XML file. Whenever any changes are made to the store hierarchy, the XML file is edited, and that file is then imported to Central Office or Returns Management. The Data Import (DIMP) Subsystem enables the importing of the store hierarchy. For information on using DIMP, see the *Oracle Retail POS Suite/Merchandising Products Implementation Guide*.

Importing Data in Returns Management

Within Returns Management, select **Data Management** to display a list of data import options. You can import data immediately or schedule an import for later. See the *Oracle Retail Returns Management User Guide*.

The following types of data can be imported:

- RM Customer Data
- Application Parameters

  Application parameters are specific to Returns Management.

Note: Application parameters are imported as part of the install process. See the *Oracle Retail Returns Management Installation Guide* for more information.

Point-of-Service Devices

Point-of-Service devices are configured with the PosDeviceTechnician.xml file, device-specific property files, and other JavaPOS configuration files. The device vendor typically provides a JavaPOS configuration file to support the JavaPOS standards. If necessary, you can create your own configuration file to meet your device requirements and replace the XML configuration file name for DeviceTechnician in ClientConduit.xml. Interaction of the Point-of-Service application with devices is managed by the Device Manager and Device Technician.

Set Up the Device

To configure a device to work with Point-of-Service, first consult the user manual for that device for specific setup requirements. Set up the device drivers and configuration file so the device is available to applications.

Test the Device

Use the POStest application available internally or at http://www.javapos.com to determine if a device adheres to existing JavaPOS standards. POStest is a GUI-based utility for exercising Point-of-Service devices using JavaPOS. Usually this requires adding the device to the jpos.xml file that is in the Point-of-Service classpath. Currently it supports the following devices: POSPrinter, MICR, MSR, Scanner, Cash Drawer, Line Display, Signature Capture, and PIN Pad. Do the following to use POStest:

1. Configure the classpath for JavaPOS. This means that the classpath should include the location of POStest, jpos.jar, jcl.jar and the JavaPOS services for the devices.
2. To build POStest, compile the classes in `<location of POStest>/upos/com/jpos/POSTest`.
3. To run POStest, enter the following at a command line:

   ```
   java com.jpos.POStest.POStest
   ```

Sometimes, the hardware vendor provides test utilities that come with the JavaPOS implementation. You should test with these tools as well.

Create a Session and ActionGroup

In Point-of-Service code, devices require a Session and an ActionGroup. If you need to interact with a new JavaPOS device, you must create a new Session and ActionGroup.

Sessions capture input for the application. In UI scripts, device connections are defined that allow the application code to receive input from a device by connecting the Session with the screen specification. The Session listens to JavaPOS controls on the device.

ActionGroups provide the actions that can be performed with the device. ActionGroups are instantiated by Tour code. When a method on an ActionGroup is called in Tour code, the DeviceTechnician talks to JavaPOS controls on the device.

To create or modify a Session and ActionGroup, perform the following steps.

1. Configure the Session and ActionGroup in `config\technician\PosDeviceTechnician.xml`

   To do this, enter the name of the Session and ActionGroup in PosDeviceTechnician.xml. You must specify the name of the object, its class and its package. In addition, you can set some attributes available in the corresponding class in PosDeviceTechnician.xml. This file creates a hash table of ActionGroups and Sessions, which are part of the DeviceTechnician. Below is a definition of an ActionGroup and Session from posdevices.xml.

Example 6–4 ActionGroup Configuration

```
<ACTIONGROUP name="LineDisplayActionGroupIfc"
             class="LineDisplayActionGroup"
             package="oracle.retail.stores.pos.device"/>
```

Example 6–5 Session Configuration

```
<SESSION name="ScannerSession"
           devicename = "defaultScanner"
           class="ScannerSession"
           package="oracle.retail.stores.foundation.manager.device"
           defaultmode = "MODE_RELEASED"
           />
```

2. Define a Session class to get input that extends InputDeviceSession or DeviceSession.

   Each type of device has a Session class defined in <*source_directory*>\applications\pos\src\oracle\retail\stores\pos\device. A device session like CashDrawerSession would extend DeviceSession, whereas an input device session like a ScannerSession would extend InputDeviceSession.

   Sessions are not instantiated in Tour code but are accessed by UI scripts in device connections.

3. Define an ActionGroupIfc interface that extends DeviceActionGroupIfc.

   This class should also be located in <*source_directory*>\applications\pos\src\oracle\retail\stores\pos\device. The following line of code shows the header of the CashDrawerActionGroupIfc class.

   ```
   public interface CashDrawerActionGroupIfc extends DeviceActionGroupIfc
   ```

4. Create the ActionGroup class. This class should be located in <*source_directory*>\applications\pos\src\oracle\retail\stores\pos\device, and its purpose is to define specific device operations available to Point-of-Service. The following line of code shows the header of the CashDrawerActionGroup class.

   ```
   public interface CashDrawerActionGroup extends CashDrawerActionGroupIfc
   ```

5. If one does not already exist, create a device connection in the UI Subsystem file. Device connections in the UI Subsystem files allow the application to receive input data from the Session.

   The DeviceSession class is referenced in the device connections for the relevant screen specifications. For example, the following code is an excerpt from <*source_directory*>\applications\pos\src\oracle\retail\stores\pos\services\tender\tenderuicfg.xml.

Example 6–6 Example of Device Connection

```
<DEVICECONNECTION
            deviceSessionName="ScannerSession"
            targetBeanSpecName="PromptAndResponsePanelSpec"
            listenerPackage="java.beans"
            listenerInterfaceName="PropertyChangeListener"
            adapterPackage="oracle.retail.stores.foundation.manager.gui"
            adapterClassName="InputDataAdapter"
            adapterParameter="setScannerData"
            activateMode="MODE_SINGLESCAN">
```

6. Access the device manager and input from the Session in the application code.

   Using the bean model, data from the Session can be accessed with methods in the device's ActionGroupIfc.

Example 6–7 ActionGroup in Tour code

```
POSDeviceActions pda = new POSDeviceActions((SessionBusIfc) bus);
pda.clearText();
pda.displayTextAt(1,0,displayLine2);
```

Simulate the Device

It is often practical to simulate devices for development purposes until the hardware is available or the software is testable. Switching to a simulated device is easily accomplished by editing `config\technician\PosDeviceTechnician.xml`. In fact, when you install Point-of-Service and choose the option to run in Simulated mode, `PosDeviceTechnician.xml` is modified accordingly. By default, unselected devices are set up as simulated. The following code samples show the difference between a normal device configuration and a simulated device configuration. Note the class name and device name are changed.

Example 6–8 Normal Device Configuration

```
<SESSION name="PrinterSession"
devicename = "defaultPrinter"
class="PrinterSession"
package="oracle.retail.stores.foundation.manager.device"
defaultmode
```

Example 6–9 Simulated Device Configuration

```
<SESSION name="SimulatedPrinterSession"
            devicename = "defaultPrinter"
            class="SimulatedPrinterSession"
            package="oracle.retail.stores.foundation.manager.device"
            defaultmode = "MODE_RELEASED"
            />
```

Scheduling Post Processors in Back Office

Schedule post processor jobs after installing Back Office. See the *Oracle Retail Back Office User Guide* for more information.

Scheduling Post Processors in Returns Management

After installation, you must schedule postprocessor jobs as part of the configuration process. Post processors create summary data for use in reporting. See the *Oracle Retail Returns Management User Guide* for more information.

Data Management in Central Office

Within Central Office, select the **Data Management** tab to display a list of data import and export options. You can import or export data immediately or schedule an import or export for later. The following types of data can be imported or exported:

- POSlog
- EJournal
- Store parameters and Central Office application parameters (see Parameters in Back Office and Central Office)

Help Files in Point-of-Service

The Point-of-Service application includes help files to provide information to assist the end-user. When the user chooses F1 from the global navigation panel, a help browser appears in Point-of-Service to describe the current screen. An index is provided on the left so the user may choose additional topics to view. The help is implemented as JavaHelp and includes these components:

- One HTML help file for each screen. The product help files are Microsoft Word files saved as HTML. They can be edited with Word, an HTML editor or a text editor.
- A Table of Contents file that defines the index that displays on the left.
- A properties file that associates overlay screen names with the corresponding HTML filenames.

For more information on JavaHelp, refer to: http://www.oracle.com/technetwork/java/javase/index-jsp-142628.html.

Note: If the base product help files are modified, upgrades for help files will not be available, and you will not be able to take advantage of updates provided with future maintenance releases of the application.

Modifying Help Files in Central Office and Back Office

Online help is created using Oracle Online Help for the Web. Information on this technology is available at: https://www.oracle.com/technology/docs/tech/java/help/index.html

The online help is generated from the application user guide. Each chapter in the user guide is divided into sections. You can look at the Table of Contents for the user guide to see how each chapter is structured. When the user guide is converted into online help, each section is converted into an html help file.

Some help files contain specific information for a screen. Other help files have the background or topic information that is contained in the user guide. For screen help, the name of the file includes the name of the screen. For background help, the name of the file is based on the section in the user guide. For example, the help file for the User Details screen is named userdetailshelp.htm. The information in the Working with Transactions section is in the workwithtransactionshelp.htm file.

For example, in Central Office, the centraloffice.ear file contains the centraloffice-help.war. The war file contains the following:

```
helpsets folder
    co_olh folder
        dcommon folder (definitions for styles, gif files for buttons)
        img folder (any images included in the online help from the user guide)
        help files
```

Note: If you have Labels and Tags installed, online help is in `lt_olh`. You will have both `bo_olh` and `lt_olh`.

To update a help file:

1. Locate the help file to be changed.
2. Edit the help file.
3. Replace the updated file in the helpset and in centraloffice.ear.
4. Redeploy centraloffice.ear or backoffice.ear.

Modifying Help Files in Point-of-Service

To modify Help Files in Point-of-Service, do the following:

1. Locate the name of the help file associated with the overlay screen name that needs to be modified. The help file names are defined in helpscreens.properties located in <*source_directory*>\applications\pos\deploy\client\config\ui\help.

Example 6–10 JavaHelp—helpscreens.properties

```
REFUND_OPTIONS                          refundoptionshelp.htm
```

2. Locate the help file in the `config\ui\help` directory. Open the file in Microsoft Word or an HTML editor and edit the content. If you are using Word to edit, be sure to save the file as HTML when the edits are complete.
3. If the index location or text descriptions needs to be modified, change toc.xml located in <*source_directory*>\applications\pos\locales\en\config\ui\help. The order of the items in the index is also defined by this file.

Example 6–11 JavaHelp—toc.xml

```
  <tocitem target="REFUND_OPTIONS"    text="Refund Options" />
```

Reason Codes in Point-of-Service

Reason codes are items offered to the end user as choices in lists, for example, the set of possible reasons for a price override. These choices normally vary for each retailer, and they must be configured to suit your local requirements and policies. The system comes with a predetermined set of reason code groups; within each group, you can add, remove, and modify the list of codes, all from within the Point-of-Service interface.

To modify reason codes:

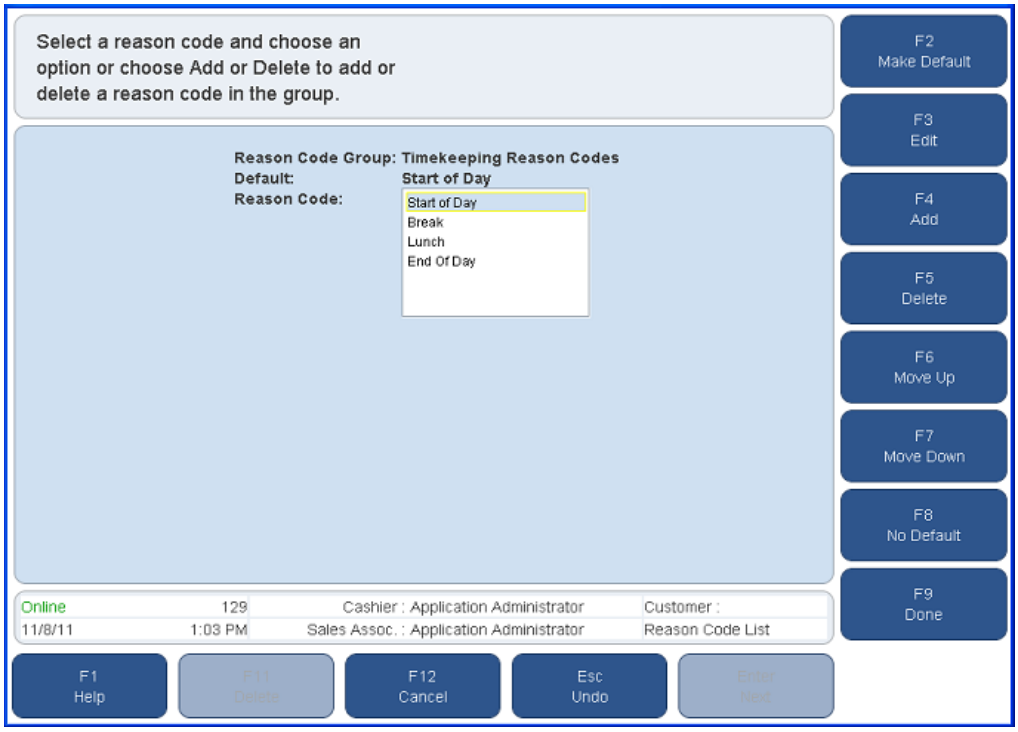
1. From the Main Options screen, press **F4/Administration**, enter the user ID and password, and press **F5/Manager**.
2. Press **F4/Security** and then **F5/Reason Codes**.

Figure 6–1 Reason Code Group Screen

3. From the Reason Code Groups screen, select the group you want to view or edit. The Reason Code List screen appears.

> **Note:** If the Edit Reason Codes parameter is set to **No**, the reason codes are for viewing only.

Figure 6–2 Reason Code List Screen

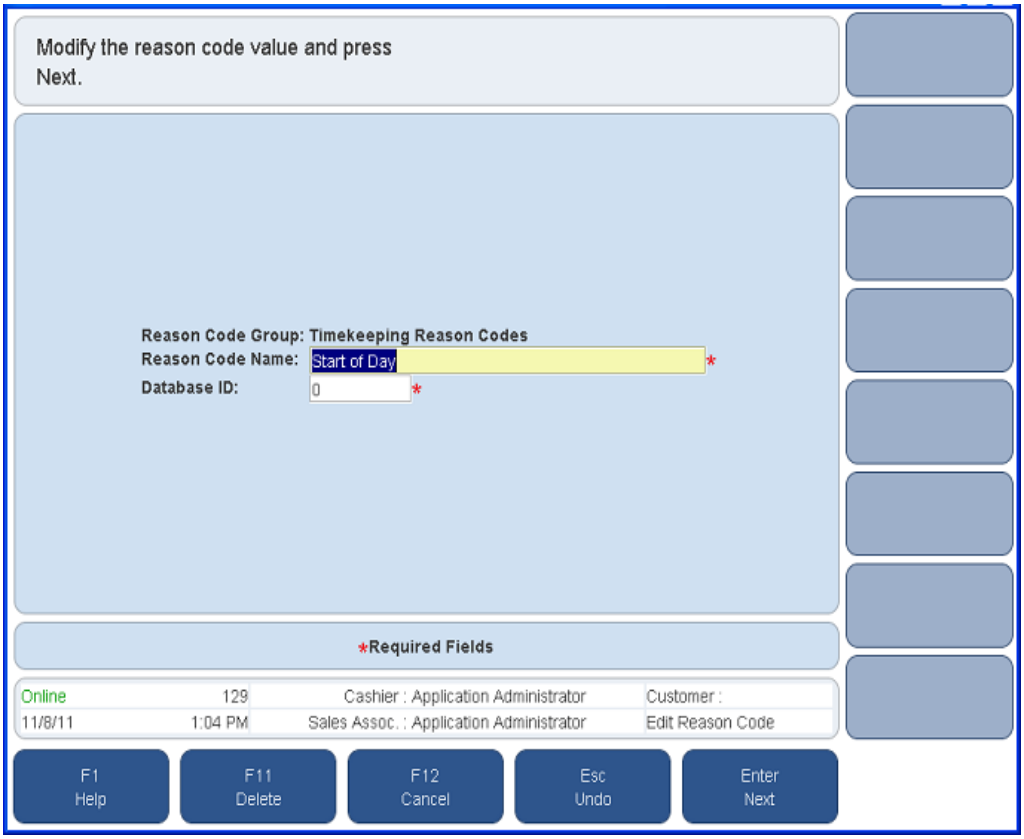


4. Select one of the following:

   - To delete a code, select it, then choose **F5/Delete**.
   - To change the position of a code in the list, select it, then choose **F6/Move Up** or **F7/Move Down**.
   - To add a code, choose **F4/Add**. The Add Reason Code screen appears. Enter a name and database ID, then choose **Enter/Next**.
   - To change the name or database ID of a code, select the code in the list and choose **F3/Edit**.

     The system displays the Edit Reason Code screen. Edit the values shown, then choose **Enter/Next**.

Figure 6–3 Edit Reason Code Screen



5. Press **F2/Make Default** to save your changes and make the selected settings the new default.
6. Choose **Enter/Next**. The changes are saved, and the system displays the Reason Code Group screen.

Configuring Transaction ID Lengths

Point-of-Service allows for some configuration of the length of the Transaction ID. These changes affect every aspect of the software and should not be undertaken lightly. Changes should only be performed before Point-of-Service is installed. Changes to these settings can require substantial custom code and testing to establish that no problems result from the change.

Note: Only the default values for these parameters were tested in the integration to Point-of-Service. Changing the values of the Point-of-Service Transaction ID settings without changing the supporting configuration for Central Office and Returns Management will cause the integration to not work correctly.

Understanding Transaction IDs

A transaction ID is a composite key made from the store number, register number, and sequence number. When combined, these attributes create a unique number for each transaction. Transaction IDs can also include an eight-digit date to ensure that they are unique. For example, if you restart your sequence numbers on a daily basis, the date value prevents transaction ID repetition.

Key points about the transaction ID and related properties:

- You can change the length of the store, register, and sequence numbers which contribute to the transaction ID. You cannot directly configure the length of the transaction ID itself.
- System-generated unique Layaway numbers, Special Order numbers, and Web Order numbers are not affected by changes to the transaction ID rules.
- A maximum of 20 digits of transaction ID can be printed on receipts using Point-of-Service current barcode format.
- If the value of a store, register, or sequence number has fewer than the specified number of digits, Point-of-Service uses leading zeroes to pad the number to the required number of digits; a four-digit sequence number whose value is 22 shows up within the transaction ID as 0022.
- Dates can be used in transaction IDs to help ensure unique IDs. If they are used, they are expressed as an 8-digit number; this is set by the `TransactionIDBarcodeDateFormat` property in the `domain.properties` file. The only valid values for this property are no value and `yyyyMMdd`. The date format does not vary from one locale to another.
- You can set the transaction sequence start number in the database.
- When you enter a transaction ID manually, the trailing date is optional.

Changing Transaction ID Format

Changing the format of the transaction ID requires many steps and requires additional testing and possibly custom code to support the merchant's desired format. See Configuring Transaction ID Lengths for more information. The base format is divided into three sections:

- Store ID
- Workstation ID
- Sequence number

See "Understanding Transaction IDs" for more information about these sections.

Example 6–12 Transaction ID Configuration in domain.properties

```
# Transaction ID
TransactionIDStoreIDLength=5
TransactionIDWorkstationIDLength=3
TransactionIDSequenceNumberLength=4
#TransactionIDBarcodeDateFormat=yyyyMMdd
TransactionIDBarcodeDateFormat=
TransactionIDSequenceNumberSkipZero=false
TransactionIDSequenceNumberMaximum=9999
```

Do the following to change the default length of the sequence number:

1. Change the length in the domain.properties in the Point-of-Service server and client:
   - TransactionIDSequenceNumberMaximum=99999
   - TransactionIDSequenceNumberLength=4
2. Update the parameters under group TransactionID in centraloffice.xml with the required length.
3. Change the regular expression format in validation.properties for TransactionNumber in Central Office. The file is in <*source_directory*>\installer\templates:

```
Validator.TransactionNumber=^[a-zA-Z0-9]{5}[a-zA-Z0-9]{3}[0-9]{4}$
```

Do the following to change the default length of the store ID and workstation ID:

1. Update the table definitions for the columns ID_STR_RT and ID_WS to the required length. By default, they are defined as:
   - store ID = 5 characters
   - workstation ID = 3 characters
2. Change the length in domain.properties in the Point-of-Service client and server:
   - TransactionIDStoreIDLength=5
   - TransactionIDWorkstationIDLength=3
3. Update the parameters under group TransactionID in centraloffice.xml with the required length.
4. Change the format in validation.properties for TransactionNumber in Central Office. The file is in <*source_directory*>\installer\templates.

> **Note:** Be sure to test the Point-of-Service, Central Office or Returns Management integration, and all screens where search by transaction ID is allowed.

Configuring the Purchase Date Field for Returns and Voids

You must configure Point-of-Service to display the Purchase Date field in the Receipt Info screen when conducting a return or a void.

To do this, you must modify the domain.properties file in the config folder. Uncomment the following field:

```
TransactionIDBarcodeDateFormat=yyyyMMdd
```

By default, this field in domain.properties contains no defined date format. This prevents the Purchase Date field from being displayed in the Receipt Info screen.

Configuring RMI Timeout Intervals in Point-of-Service

You can configure remote method invocation (RMI) timeout intervals at two levels:

- The JVM level (Linux installs only)
- The level of managers and technicians

If you are performing a Linux installation, configure the JVM as described in "Setting the RMI Timeout Interval for the JVM Under Linux", below. If you determine that RMI connections are timing out, you can use one of the other procedures in this section, "Setting the RMI Timeout Interval for All Manager and Technician Calls" or "Setting the RMI Timeout Interval for a Specific Technician".

Setting the RMI Timeout Interval for the JVM Under Linux

Oracle Retail has found it useful to change the RMI timeout interval for the JVM under Linux. To do this, change the command that launches the JVM, adding the JVM flag: `Dsun.rmi.transport.connectionTimeout=<X>` where `<X>` represents the time-out period in milliseconds.

This tells the JVM to time out socket connections used by RMI after X milliseconds of inactivity. Linux quickly notifies the JVM when a socket connection cannot be established. Linux is slow, however, to notify the JVM when an open socket connection has been broken. By setting the connection time-out low, you can cause the sockets to disconnect quickly after each RMI call, thereby requiring a connect for each subsequent RMI call.

Modifying the TCP Connection Timeout on Linux

Sometimes, Linux keeps the tcp connection active even after Point-of-Service determines that the socket has timed out. There are three OS level settings that work together to determine how long to keep the tcp connection open, which affects the observed system performance. To modify these level settings, at a Linux command line, enter:

```
sysctl -w net.ipv4.tcp_keepalive_time=<value>
sysctl -w net.ipv4.tcp_keepalive_intvl=<value>
sysctl -w net.ipv4.tcp_keepalive_probes=<value>
```

where *<value>* is an interval you specify.

Setting the RMI Timeout Interval for All Manager and Technician Calls

You can change the RMI timeout interval values for connections and reads in the *<source_directory>*`\applications\pos\deploy\`*<Client or Server>*`\bin\comm.properties` file. The value for the following properties apply to all manager and technician calls, unless overridden by a communication scheme for a specific call.

- `comm.socket.connectTimeout` - Specifies how long to wait for a socket connection to succeed. The value is in milliseconds.
- `comm.socket.readTimeout` - Specifies how long to wait before a read times out. The value is in milliseconds. This property causes the read to time out even if the socket is alive and well and transmitting data.

Note: These values control the application timeout when trying to establish a socket connection or read from a socket.

Setting Application Timeout Values on Linux

Do the following when configuring the application timeout values for Point-of-Service on Linux:

1. Set the socket timeout values in the `comm.properties` file:

   ```
   comm.socket.readTimeout=25000
   comm.socket.connectTimeout=25000
   ```

2. Set the RMI property values in the startup script, for example, in `ClientConduit.sh`:

   ```
   JAVA_OPTIONS=${JAVA_OPTIONS}"-Dsun.rmi.transport.tcp.responseTimeout=5000"
   ```

   Other possible values include the following:

   ```
   -Dsun.rmi.transport.tcp.logLevel=VERBOSE
   -Dsun.rmi.transport.tcp.responseTimeout=5000
   -Dsun.rmi.transport.logLevel=VERBOSE
   -Dsun.rmi.transport.tcp.readTimeout=1500
   -Dsun.rmi.transport.tcp.handshakeTimeout=5000
   -Dsun.rmi.transport.proxy.connectTimeout=10000
   -Dsun.rmi.transport.connectionTimeout=15000
   ```

   These values are described at http://java.sun.com/j2se/1.5.0/docs/guide/rmi/sunrmiproperties.html.

3. Set the Linux tcp property values.

   There are three operating system-level settings that work together to determine how long to keep the tcp connection open, which affects the observed system performance. At the Linux command line, type the following:

   ```
   sysctl -w net.ipv4.tcp_keepalive_time=<value>
   sysctl -w net.ipv4.tcp_keepalive_intvl=<value>
   sysctl -w net.ipv4.tcp_keepalive_probes=<value>
   ```

   Additional information can be found at http://ibdeveloper.com/issues/issue-1-sep-1-2005/using-keepalive-sockets-to-detect-and-release-hung/.

Setting the RMI Timeout Interval for a Specific Technician

To set the time-out for a specific technician, edit the *<source_directory>*`\applications\pos\deploy\<`*Client or Server*`>\bin\comm.properties` file and the conduit script as follows:

1. Add a new communication scheme to the `comm.properties` file. The following lines provide an example:

   ```
   comm.rmi_longread.readTimeout=120000
   comm.rmi_longread.connectTimeout=1000
   ```

   These lines establish a new communication scheme called rmi_longread with a read time-out of 120 seconds and a connect time-out of one second (since the values are in milliseconds).

2. Add the following property to the appropriate technician definition in the conduit script:

   ```
   <PROPERTY propname="commScheme" propvalue="rmi_longread"/>
   ```

This sets the communication time-outs for all managers that connect to this technician. A manager who is sending a valet to this technician times out if the valet fails to complete within 120 seconds. It only attempts to connect to the technician for 1 second before giving up.

System Settings in Point-of-Service

System settings are values set in the Oracle Retail database. Changes to these settings must be made in the database by a database administrator or an application developer.

System settings can have significant effects on Point-of-Service system; do not make changes unless you are confident that you understand the effects. For a description of all available system settings, refer to the *Oracle Retail POS Suite Configuration Guide*.

Configuring Logging in Point-of-Service

Point-of-Service logging uses the Log4J tool. Configure Log4J by editing `<source_directory>\applications\pos\deploy\shared\config\log4j.xml`. See the Apache documentation for Log4J at `http://logging.apache.org/log4j`. For more information, a Log4j XML Configuration Primer can be found at `http://wiki.apache.org/logging-log4j/Log4jXmlFormat`.

Returns Management Environment Entries in ejb-jar.xml

This section describes the *<env-entry>* section in the Returns Management ejb-jar.xml file. These entries enable manipulation of some aspects of the system.

Return Ticket Formatting Entries

The return ticket table is indexed using a composite key. This key is comprised of store number, workstation, business date, and a sequence number. To make this key end-user legible, it is formatted using the returnTicketIdPattern rather than passed as discrete data elements. The default pattern is *sssss-www-MMdd-yyyy-nnnnnnnn*.

Table 6–3 Return Ticket Table

Data Element	Description
sssss	Marks the store ID.
www	Marks the workstation ID.
nnnnnnnn	Marks the sequence number.
MMdd-yyyy	Marks the business date.

This *<env-entry>* element must be in sync with the other *<env-entry>* elements as follows:

- ID divider: If you want to use a different divider, then the value returnTicketIdDivider must be changed to reflect the new divider used.
- Date format: If this format is changed, other than the divider character, the value returnTicketIdDatePattern must be changed to reflect this.
- Store pattern: If the store retrieved from the database is shorter than the value in returnTicketStoreIdPersistPattern, it is padded on the left hand side with the value in returnTicketPersistPad (default is **0**).

- Sequence pattern: If the sequence number is smaller than the length of returnTicketSeqNumberPersistPattern, then it is padded with the value from returnTicketIdPad (default is **0**).
- ReturnTicketMaxSequenceValue: No effect.
- ReturnTicketBusinessDate: Ignore. This is used in an unused method in the ReturnTicketKeyFormatter class and can be safely ignored.

Table 6–4 defines the return ticket format elements.

Table 6–4 Return Ticket Format <env-entry>

Entry	Default
returnTicketIdPattern	sssss-www-MMdd-yyyy-nnnnnnnnn
returnTicketIdDivider	NA
returnTicketIdDatePattern	MMddyyyy
returnTicketStoreIdPersistPattern	sssss
returnTicketWorkstationIdPattern	www
returnTicketSeqNumberPersistPattern	nnnnnnnnn
returnTicketIdPad	0
returnTicketPersistPad	0
returnTicketMaxSequenceValue	999999999
returnTicketBusinessDate	yyyy-MM-dd

Auditing Entries

Table 6–5 identifies audit target format elements.

Table 6–5 Audit Target <env-entry>

Entry	Default
journalDataPath	1

This integer value tells Returns Management where to send audit log messages. Valid values are:

- 0 – no audit log
- 1 – send log messages to a JMS queue (found using JNDI lookup at java:comp/env/jms/JournalingMessage)
- 2 – send log messages directly to the EJB interface of the Journaling Service.

Any other value results in no audit log being created and an error message logged.

Configuring Security in Returns Management

Returns Management has many individual security access points. This enables you to control the functionality to which any particular end user has access. You can also control workflow through approval permissions, enabling some employees to schedule tasks which others must approve.

For more information about security roles, see the *Oracle Retail Returns Management User Guide*.

7

Oracle Retail Returns Management Integration Methods and Communication Flow

The main integration point of Returns Management is with an external point-of-return. To communicate between the systems, Returns Management provides methods which accept and return messages in a predefined format. This chapter discusses the methods and the messages. This chapter also discusses some of the implications of the chosen implementations.

Methods of Contact

Returns Management has two primary methods of contact with the point-of-return:

- The point-of-return requests return authorization from Returns Management (evaluation).
- The point-of-return notifies Returns Management of what was actually returned (exception tracking).

Both of these methods use XML messages. The call to evaluation is a synchronous call that returns a separate XML message. The call to scoring is an asynchronous call.

Returns Management Messages

The three messages defined by Returns Management are:

- Return Request
- Return Response
- Return Result

The return request is passed from the point-of-return to Returns Management when evaluation is invoked. The Return Response is returned by Returns Management to show the result of evaluation. The return result is passed by the point-of-return to Returns Management to initiate scoring.

This section describes the integration of Returns Management with an external point-of-return, using an example transaction and sample XML messages that are sent between the point-of-return and Returns Management.

To more clearly illustrate the XML messages, this chapter provides a scenario of a customer returning items under different situations. Each situation has a sample of the XML message with details around each element in the XML. There is a sample XML file for each of the three basic messages:

- Return Request
- Return Response
- Return Result

Each of these messages has a corresponding XSD that defines the valid XML for each message.

Sample XML for Return Transaction Scenarios

John Smith wants to return some sporting equipment he has purchased. We will examine the message sent from the point-of-return to Returns Management when he first wants to return the items. Returns Management will respond asking for positive ID. A second return request is made from the point-of-return to Returns Management with the additional ID information. Returns Management then responds with its decision.

The customer decides he wants to return the items. Then, for whatever reason, the return is voided. Finally, the customer decides to re-return the items when Returns Management is offline.

All XSD's referenced are provided in the Returns Management installation material. Table 7–1 identifies XSD file locations within the EPD install package.

Table 7–1 **XSD Locations**

Document Name	Location
Return Request	returnsmgmt/api/returnsSchemas.zip/RM-ReturnRequest.xsd
Return Response	returnsmgmt/api/returnsSchemas.zip/RM-ReturnResponse.xsd
Return Result	returnsmgmt/api/returnsSchemas.zip/RM-ReturnResult.xsd

Point of Return to Returns Management—Initial Return Request

In this scenario, John Smith has decided he wants to return some baseballs. He goes to the point-of-return which emits the following message:

Example 7–1 Initial Return Request

```
<ReturnRequest>
    <itemReturnInfo>
        <itemTransactionInfo>
            <receipted>true</receipted>
            <transactionID>
                <storeID>12345</storeID>
                <workstationID>124</workstationID>
                <sequenceNumber>2</sequenceNumber>
                <businessDate>2005-12-31</businessDate>
            </transactionID>
            <found>true</found>
            <validAtPointOfReturn>true</validAtPointOfReturn>
            <giftReceipt>false</giftReceipt>
            <purchaseDate>2005-12-31</purchaseDate>
            <deliveryDate>2006-01-01</deliveryDate>
```

```
            <validationAmount>40.00</validationAmount>
            <originalTender>
                <tenderType>
                    <type>CASH</type>
                    <amount>12.00</amount>
                </tenderType>
                <tenderType>
                    <type>CRDT</type>
                    <amount>38.00</amount>
                    <cardNumber>1234567812345678</cardNumber>
                </tenderType>
            </originalTender>
            <saleQuantity>10</saleQuantity>
        </itemTransactionInfo>
        <itemIdentifier>
            <itemID>40020002</itemID>
            <itemDescription>MLB Baseball</itemDescription>
            <itemType>Sporting Good</itemType>
        </itemIdentifier>
        <returnReason>Customer Satisfaction</returnReason>
        <quantity>10.00</quantity>
        <amountPaidPerUnit>4.00</amountPaidPerUnit>
        <requestedAdjustedPrice>4.00</requestedAdjustedPrice>
        <itemCondition>New</itemCondition>
        <manuallyEntered>false</manuallyEntered>
    </itemReturnInfo>
    <returnStoreID>04241</returnStoreID>
    <returnWorkstationID>123</returnWorkstationID>
    <employeeID>20051</employeeID>
    <customerInfo>
        <customerID>80012</customerID>
    </customerInfo>
    <transactionType>Return</transactionType>
</ReturnRequest>
```

The entire request has a root element of **<ReturnRequest>**.

The following sub- elements, unless specified otherwise, are of type String and are required.

<itemReturnInfo> complex type
Each return request is based around returning a discrete number of items. Each unique type of item has a corresponding itemReturnInfo element. This means that if a customer is returning ten baseballs and two bats, then there are two itemReturnInfo elements, not twelve.

<itemTransactionInfo> complex type, sub-element of <itemReturnInfo>
This complex type describes the transaction during which the item was originally purchased.

<receipted> Boolean, sub-element of <itemTransactionInfo>
This Boolean element tells Returns Management whether the customer has a receipt for this item. Non-receipted returns are allowed, but can trigger a different return policy.

<transactionID>, optional, complex type, sub-element of <itemTransactionInfo>
This complex type identifies the ARTS-compliant transaction of the original purchase, if any.

<storeID>, <workstationID>, <sequenceNumber>, <businessDate>, sub-elements of <transactionID>
These elements correspond to the parts of the ARTS-compliant transaction ID.

Note:

- The sequence number is a positive integer and the business date is a date.
- The store ID and workstation ID of these elements do not need to match the store ID and workstation ID of <returnStoreID> and <returnWorkstationID>. Therefore, the item can be purchased at one store and returned at another.

<found> Boolean, sub-element of <itemTransactionInfo>
This Boolean element tells Returns Management if the transaction ID from the <transactionID> element was found. This element exists because it is possible to have a transaction number, for example, from a receipt, that is not found by the point-of-return when it queries existing transaction data. This element is required, but is only relevant if there is a transaction ID. If there is no <transactionID> element, this value should be set to **false**.

<validAtPointOfReturn> Boolean, sub-element of <itemTransactionInfo>
This Boolean element tells Returns Management if the transaction ID from the <transactionID> element is considered valid by the point-of-return. Any transaction ID that is found should set this value to **true**. If the ID is not found, the point-of-return should decide if the transaction ID appears to be legitimate and set this value accordingly.

<giftReceipt> optional, Boolean, sub-element of <itemTransactionInfo>
This Boolean element tells Returns Management if the receipt presented at the point-of-return is a gift receipt. This element should not be included in the message if the <receipted> element is **false**.

<purchaseDate>, <deliveryDate>, optional, date, sub-element of <itemTransactionInfo>
These date elements refer to the purchase and delivery dates of the item being returned, respectively. If this data cannot be determined at the point-of-return, these elements should be omitted.

<validationAmount>, optional, decimal, sub-element of <itemTransactionInfo>
This optional element represents the dollar amount of the items being returned. This is only included when there is no found transaction but there is a valid transaction ID. This could happen if, for example, a customer has a receipt with a transaction number and amount on it, but the point-of-return cannot find the transaction in storage.

<originalTender>, complex type, sub-element of <itemTransactionInfo>
This complex type represents the original tenders used to purchase these items. Though this type is required, the list of original tenders can be empty, for example, for a non-receipted, non-transaction ID return.

<tenderType>, optional, complex type, sub-element of <originalTender>
For each original tender that the point-of-return knows about, there is a tender type entry.

Note: There might be no tender type entries (for example, for non-receipted returns), one entry, or many entries (for a split-tender scenario, such as an item that was bought partially with a gift card and partially with cash).

<type>, <amount>, <cardNumber>, sub-elements of <tenderType>
These three elements describe the tender used. The <type> element is the only required element and is expected to match the standard four letter Oracle Retail tender types, for example, CASH. The <amount> element is an optional decimal value. The <cardNumber> element is listed as optional, but should be filled in with the appropriate card number if the tender type is CRDT.

<saleQuantity>, optional, decimal, sub-element of <itemTransactionInfo>
This element represents the original quantity of items sold to the customer. A customer might want to return only one out of ten items they have bought. This original sale quantity is compared to previous returns Returns Management knows about. If the sum of items from previous returns plus the items from this return is greater than the sale quantity, Returns Management can flag and deny this return attempt.

<itemIdentifier> sub-element of <itemReturnInfo>
This complex type identifies which item is being returned.

<itemID>, <itemDescription> sub-elements of <itemIdentifier>
Returns Management relies on the <itemID> when referring to an item that it can look up in the AS_ITM table of the Oracle Retail data model. Though the XSD enables <itemDescription> to be included, it is unused by the code.

<itemType> optional, sub-element of <itemIdentifier>
The <itemType> element describes the type of the item as evaluated by ItemTypeEvaluator. The user interface uses the ItemTypes parameter. The point-of-return and Returns Management must agree on valid values for this element.

<returnReason>, sub-element of <itemReturnInfo>
The reason for which the item is being returned. This required element is used by the class ReturnReasonEvaluator. Based on the text provided here, the evaluator can choose various responses during policy initiation. The user interface uses the ReturnReasons parameter. The point-of-return and Returns Management must agree on valid values for this element.

<quantity>, decimal, sub-element of <itemReturnInfo>
This is the quantity being returned. Non-unitary units of measure, for example, feet, should be expressed in a decimal format, such as 1.5 feet for 18 inches.

<amountPaidPerUnit>, decimal, optional, sub-element of <itemReturnInfo>
The amount paid per item on this return.

<serialNumber>, optional, sub-element of <itemReturnInfo>
This element is currently unused by Returns Management.

<requestedAdjustedPrice>, optional, decimal, sub-element of <itemReturnInfo>
This element is set to the price at which the point-of-return wants to return the item. For instance, the point-of-return might request to return the item for less than the original sales price. This value is compared to the original price in PriceAdjustmentAmountEvaluator class. That rule initiates different actions depending on the ratio of the adjusted price to the original price per unit.

<itemCondition>, optional, sub-element of <itemReturnInfo>
This element reflects the condition of the item. This value is used by the ItemConditionEvaluator class. Like the <returnReason> element, the legal values for this element need to be agreed upon by the point-of-return and Returns Management. The user interface uses the ItemConditions parameter. The point-of-return and Returns Management must agree on valid values for this element.

<manuallyEntered>, Boolean, sub-element of <itemReturnInfo>
This required Boolean element denotes if the information in the <transactionID> element was manually entered at the point-of-return. This element should be **false** if there is no transaction ID.

<returnStoreID>, <returnWorkstationID>, <employeeID>
These are the IDs of the store, workstation, and employee that are initiating the return, respectively. The employee ID is used for tracking cashier exceptions and is expected to correspond to an entry into the Oracle Retail employee table.

<customerType>, optional
This element type is optional for loyalty. This value is used by the CustomerTypeEvaluator class.

<customerInfo>, <moreCustomerInfo>
These complex element types represent information about the customer returning the items.

Note: Only one of these info types can be present in the return request.

<customerID>, sub-element of <customerInfo>
The <customerID> element corresponds to the Oracle Retail customer ID.

Note: This element is different from the Returns Customer ID in the Returns Management customer table, which is keyed off of positive ID.

<transactionType>
The <transactionType> element corresponds to the type of return requested by the point-of-return. Valid values are defined by the parameter RefundTypes. The point-of-return and Returns Management must agree on valid values for this element. This parameter is defined in `returnsmgmt.xml`. Default values are:

- Return
- Layaway_Cancellation
- Order_Cancellation
- Price_Adjustment

Returns Management to Point of Return—Initial Return Response: Need Positive ID

After the initial return request has been submitted, Returns Management determines that it needs a positive ID from the customer. Returns Management responds, indicating that the point-of-return should obtain the ID from Mr. Smith.

Note: The following positive ID types are supported in the base integration between the point-of-return and Returns Management:

- Driver's License
- Passport
- Military ID
- State/Region ID

Any other positive ID types set up in the point-of-return are not supported in the base integration between the point-of-return and Returns Management.

Example 7–2 Return Response Requesting Positive ID

```
<ReturnResponse>
    <returnTicketID>04241-123-1025-2006-005021791</returnTicketID>
    <responseApproveDenyCode>Denial</responseApproveDenyCode>
    <languagePreference>en</languagePreference>
    <itemReturnResponse>
        <itemIdentifier>
            <itemID>40020002</itemID>
        </itemIdentifier>
        <responseCode>10</responseCode>
        <approveDenyCode>Denial</approveDenyCode>
        <responseDescription>Insufficient quantity</responseDescription>
        <refundTenders/>
        <customerInfoRequired>true</customerInfoRequired>
    </itemReturnResponse>
</ReturnResponse>
```

The entire request has a root element of **<ReturnResponse>**.

The following sub- elements, unless specified otherwise, are of type String and are required.

<returnTicketID>
This element refers to the return ticket created by Returns Management. The point-of-return needs to use this element in future communications with Returns Management about this return (for example, in response to a request for positive ID or when sending a final result).

<responseApproveDenyCode>
If there are multiple item responses, then the most cautious <approveDenyCode> value is used.

There are exceptions to the behavior of this element. If a current entry is found in the customer service override table (RM_CT_SV_ORD) that matches this Returns Management customer and is active for the same date as the return, and the response would be a denial, then this value is set to **Approved** and the optional <availableCustomerServiceOverride> element is set.

<languagePreference>
This element informs the client in which language the <responseDescription>, <receiptMessage>, and other elements are sent. Currently this is always **en** for English.

<itemReturnResponse>, complex type
This element contains the detailed information about each of the items to which Returns Management is responding. There can be many of these elements in a transaction.

<itemIdentifier>, complex type, sub-element of <itemReturnResponse>
This complex type identifies which item is being returned.

Note: Returns Management sets the <itemID> sub-element only.

<responseCode>, <approveDenyCode>, <responseDescription> sub-elements of <itemReturnResponse>
Returns Management has a response associated with each item. These response codes are configured by the rule actions of the policy that Returns Management chose to execute. The codes are contained in the table RM_RSPS_RC.

The three values here correspond to the ID_RPSS_RC, TY_RSPS, and DE_RSPS respectively. The <responseCode> element itself is an integer that corresponds to an ID of a response code. The <approveDenyCode> is one of **Denial**, **Mgr Overridable Denial**, **Contingent Authorization**, or **Authorization**. The <responseDescription> is the short description of the response. Though these fields are required, in the message they can be ignored by the point-of-return since the <customerInfoRequired> element is **true**.

<customerInfoRequired>, optional, Boolean, sub-element of <itemReturnResponse>
If this value is set to **true**, Returns Management is asking the point-of-return to prompt for Encrypted Positive ID. If this value is not present the point-of-return should assume that it is **false**.

Point of Return to Returns Management—Second Return Request

Once the point-of-return has gotten a positive ID for Mr. Smith, it returns that information to Returns Management along with the data from the original return request.

Note: The <itemreturnInfo> content is the same as in Example 7–1, "Initial Return Request" and has been left out for brevity.

Example 7–3 Second Return Request

```
<ReturnRequest>
    <itemReturnInfo>
        ...
    </itemReturnInfo>
    <returnStoreID>04241</returnStoreID>
    <returnWorkstationID>123</returnWorkstationID>
    <employeeID>20051</employeeID>
    <moreCustomerInfo>
        <lastName>Smith</lastName>
        <firstName>Carlos</firstName>
        <middleName>Juan</middleName>
        <gender>Male</gender>
        <birthDate>1972-06-25</birthDate>
        <address1>1234 Example Blvd</address1>
        <address2/>
        <city>Miami</city>
        <state>FL</state>
        <postalCode>33056</postalCode>
        <country>United States</country>
        <countryCode>US</countryCode>
        <telephoneAreaCode>888</telephoneAreaCode>
        <telephoneLocalNumber>5551212</telephoneLocalNumber>
    </moreCustomerInfo>
    <positiveID>
        <EncryptedPositiveID>12345678</EncryptedPositiveID>
        <type>DriversLicense</type>
        <issuer>US_MN</issuer>
        <issued>2004-01-01</issued>
        <expiration>2007-01-01</expiration>
    </positiveID>
    <transactionType>Return</transactionType>
    <returnTicketID>04241-123-1025-2006-005021791</returnTicketID>
</ReturnRequest>
```

The entire request has a root element of **<ReturnRequest>**.

Returns Management works with two different customer data stores. The first data store is the Returns Management customer data store, which is keyed off of positive IDs. This is the set of customers used for exception tracking. The second data store is the standard Oracle Retail customer data store. Returns Management will attempt to link customers for which it has a positive ID to customers in the Oracle Retail customer data store, creating customers if necessary.

For purposes of the Returns Management customer, only the <positiveID> element is relevant. Returns Management either looks up or creates the customer corresponding to the positive ID. For the Oracle Retail customer, there are two elements which matter: <customerID> (underneath <customerInfo>) and <moreCustomerInfo>. If the <customerID> is passed in, Returns Management assumes that this is the Oracle Retail customer relevant to the message. If the <customerID> element is absent, and both the <positiveID> and the <moreCustomerInfo> elements are present, then Returns Management not only looks up or creates the Returns Management customer, but it also creates a new Oracle Retail customer and associates it with the Returns Management customer.

These examples are contrived to display the <customerInfo> and <moreCustomerInfo> elements. In the first message, the point-of-return had a valid customer ID. In this message, the XML is constructed to imply that Returns Management should create a new customer matching the positive ID. In a real usage scenario, if the point-of-return knew the Oracle Retail customer but just needed to collect positive ID, the point-of-return would send the <customerInfo> element again rather than the <moreCustomerInfo> element.

<moreCustomerInfo>, optional, complex type
This element contains the information necessary to create an Oracle Retail customer.

<lastName>, <firstName>, sub-elements of <moreCustomerInfo>
These elements contain the last and first names (respectively) of the new Oracle Retail customer.

<middleName>, <gender>, <birthDate>, optional, sub-elements of <moreCustomerInfo>
These optional elements contain the middle name, the gender, and the birth date of the new Oracle Retail customer. Notice that these elements are all strings, including birth date. Also notice that the gender element is constrained to either male or female.

<address1>, <address2> sub-elements of <moreCustomerInfo>
These elements correspond to the usual two address lines of a customer. In this example, though <address2> is present, it is blank.

<city>, <state>, <postalCode>, sub-elements of <moreCustomerInfo>
These elements are further parts of the customer address. In the US, the state and postal code would correspond to the state and zip code. In Canada, they would correspond to the province and postal code.

<country>, optional, sub-element of <moreCustomerInfo>
This element corresponds to the country in which the customer resides.

<telephoneAreaCode>, <telephoneLocalNumber>, optional, sub-elements of <moreCustomerInfo>
These two optional elements reflect the telephone number and area code. For a number such as “888-555-1212”, the “888” would be in the <telephoneAreaCode> element while the “5551212” would be in the in the <telephoneLocalNumber> element.

<positiveID>, complex type
Positive ID refers to a customer presenting credentials (such as a driver’s license) to authenticate their identity. The positive ID is used to reference the Returns Management customer. This element encodes information about the type of positive ID gathered by the point-of-return. See above for a discussion of the Returns Management customer versus the Oracle Retail customer.

<EncryptedPositiveID>, sub-element of <positiveID>
The unique identifier on the positive ID.

<type>, sub-element of <positiveID>
The type of identification presented. Valid types are DriversLicense, MilitaryID, Passport, and StateCard.

<issuer>, sub-element of <positiveID>
This is the issuing authority of the identification. For StateCard, this is the state which issued the card. For Passport, this is the country which issued the passport.

<issued>, <expiration>, optional, date, sub-elements of <positiveID>
These two optional date elements reflect the date of issue and the date of expiration, respectively, of the positive ID.

<transactionType>
This element is the same as the element in the initial return request.

<returnTicketID>, optional
By setting the <returnTicketID> element, the point-of-return lets Returns Management know that it is responding to a request for more information. The element should be set to the ticket ID sent from Returns Management in the previous return response. In this case, the element has been set to 04241-123-1025-2006-005021791 to match our previous message.

Returns Management to Point of Return—Second Return Response

Now that Returns Management has obtained a positive ID, it can tell the point-of-return about its decision of whether to allow the return of the items.

In this scenario, Mr. Smith has been returning a lot of items lately and has had an entry put into the exception file. This would normally result in a denial. However, Mr. Smith's agent has called the customer service center and asked them to accept the return. They have entered an entry into the customer service override table for Mr. Smith. Returns Management checks for these entries while it creates the final response. In this case, since it has found one, Returns Management authorizes the return but marks it as using a customer override.

Example 7–4 Second Return Response

```
<ReturnResponse>
    <returnTicketID>04241-123-1025-2006-016085229</returnTicketID>
    <responseApproveDenyCode>Authorization</responseApproveDenyCode>
    <availableCustomerServiceOverride>true</availableCustomerServiceOverride>
    <receiptMessageNumber>1</receiptMessageNumber>
    <receiptMessageDescription>Thank you for shopping!</receiptMessageDescription>
    <languagePreference>en</languagePreference>
    <itemReturnResponse>
        <itemIdentifier>
            <itemID>40020002</itemID>
        </itemIdentifier>
        <responseCode>150</responseCode>
        <approveDenyCode>Mgr Overridable Denial</approveDenyCode>
        <responseDescription>Exception file match</responseDescription>
        <receiptMessageNumber>1</receiptMessageNumber>
        <receiptMessageDescription>Thank you for
shopping!</receiptMessageDescription>
        <refundTenders/>
    </itemReturnResponse>
</ReturnResponse>
```

Note: The return ticket ID in this response is different from the one sent in the first response. Each new return request generates a new return ticket ID.

<availableCustomerServiceOverride>, optional, Boolean
This optional Boolean element is only set when:

- The overall approve or denial code is a denial.
- This Returns Management customer has an active entry in the customer service override table.

When this item is present, it is always **true**. For details about the approve and deny code process, refer to the "<responseApproveDenyCode>" on page 7 under the initial return response.

<receiptMessageNumber>, positive integer
Returns Management has a number of receipt messages that it can send back to the point-of-return. These messages are intended to be printed on the receipt, but obviously the point-of-return can do what it would like with them. Each message has both a number and a description. The number is provided for internationalization purposes and the message is provided to make the XML more human readable.

Note that there is both a receipt message associated with both the overall return response as well as with each item on the response. The overall message is determined in the same manner as the overall response code. That is, the most cautious individual response code determines both the overall response as well as the overall receipt message.

<receiptMessageDescription>
This is the text associated with the receipt message number. This text will be in the language of the <languagePreference> element. Currently, this text is always in English.

<languagePreference>
This is the same element as detailed in the initial return response.

<itemReturnResponse>, complex type
This is the same element as returned in the first return response. In this case, the sub-elements of this complex type contain detailed information about the decision of Returns Management regarding the returnability of this item.

<approvedQuantity>, decimal, optional sub-element of <itemReturnResponse>
This element is currently unused.

<receiptMessageNumber>, sub-element of <itemReturnResponse>
This is the receipt message number associated with the individual item. See "<receiptMessageNumber>, positive integer" on previous page.

<itemDispositionCode>, optional, sub-element of <itemReturnResponse>
This element corresponds to the disposition of the item after it has been returned, for example, "keep frozen". It corresponds to the table in ID_DPSN_CD in the ARTS schema. However, this element is currently not set.

<receiptMessageDescription>, sub-element of <itemReturnResponse>
This is the receipt message text associated with the individual item. See "<receiptMessageDescription>" on previous page.

<restockingFee>, optional, decimal, sub-element of <itemReturnResponse>
This element is currently unused.

Point of Return to Returns Management—Return Result from Second Response

Once the positive ID has been collected and Returns Management has told the point-of-return about the returnability of the item, the point-of-return processes the return as necessary. Once the return has been completed, the point-of-return sends Returns Management a return result message.

Example 7–5 Return Result

```
<ReturnResult>
    <returnTicketID>04241-123-1025-2006-016085229</returnTicketID>
    <returnTransactionID>
        <storeID>04241</storeID>
        <workstationID>123</workstationID>
        <sequenceNumber>250</sequenceNumber>
        <businessDate>2006-10-25</businessDate>
    </returnTransactionID>
    <itemReturnResult>
        <itemIdentifier>
            <itemID>40020002</itemID>
        </itemIdentifier>
        <quantityReturned>10</quantityReturned>
        <finalResultCode>Authorized</finalResultCode>
        <overrideInfo>
            <managerID>20008</managerID>
            <overrideObtained>true</overrideObtained>
            <tenderOverride>false</tenderOverride>
        </overrideInfo>
        <originalTransactionID>
            <storeID>12345</storeID>
            <workstationID>124</workstationID>
            <sequenceNumber>2</sequenceNumber>
            <businessDate>2005-12-31</businessDate>
        </originalTransactionID>
        <returnTender>
            <tenderType>
                <type>CASH</type>
                <amount>40.00</amount>
            </tenderType>
        </returnTender>
    </itemReturnResult>
</ReturnResult>
```

Once again, note the return ticket ID. It is the ticket ID referring to the second return response.

<returnTransactionID>, optional, complex type
This element refers to the ARTS-compliant return transaction generated by the point-of-return. It has the same format as the <transactionID> element of the return request.

<itemReturnResult>, complex type
This complex element represents detailed information about each item returned.

<quantityReturned>, sub-element of <itemReturnResult>
This number reflects the quantity actually returned by the point-of-return.

<finalResultCode>, sub-element of <itemReturnResult>
This element has one of two values, **Authorized** or **Denial**. For items that are returned, the value is set to **Authorized**.

<overrrideInfo>, optional, complex type, sub-element of <itemReturnResult>
This complex type is included in the result if the point-of-return decided to override a return decision rendered either by Returns Management or locally.

<managerID>, sub-element of <overrideInfo>
The ID (from the ARTS-compliant table PA_EM) that corresponds to the employee ID of the manager who overrode the return decision.

<overrideObtained>, Boolean, sub-element of <overrideInfo>
This element is set to **true** if the override was about the returnability of the item.

<tenderOverride>, Boolean, sub-element of <overrideInfo>
This element is set to **true** if the override was about which tenders to return money on.

<originalTransactionID>, optional, complex type, sub-element of <itemReturnResult>
This element refers to the ARTS-compliant transaction associated with the original sale. This element has the same format as the <transactionID> element of the return request. This element is currently unused by Returns Management.

<returnTender>, complex type
This element is a list of tenders. It describes the number of tenders, and the amount of each one, used by the point-of-return to return money to the client. It has the same format as the <originalTender> element of the return request.

Point of Return to Returns Management—Void Return

Mr. Smith has proven to be indecisive and decides that he wants to have his baseballs after all. He wants to void the return and receive the items back. To accommodate this, the point-of-return voids the previous return and informs Returns Management of the fact.

Example 7–6 Void Return Result

```
<ReturnResult>
    <returnTicketID>04241-123-1025-2006-016085229</returnTicketID>
    <returnTransactionID>
        <storeID>04241</storeID>
        <workstationID>123</workstationID>
        <sequenceNumber>250</sequenceNumber>
        <businessDate>2006-10-25</businessDate>
    </returnTransactionID>
    <returnVoided>true</returnVoided>
</ReturnResult>
```

In this message, we see the same return ticket ID and the same return transaction ID of the return result above. This is used to let Returns Management know which return is being voided. The only new element is the <returnVoided> element.

<returnVoided>, optional, Boolean
This element shows that the return referenced by the <returnTicketID> element has been voided. Though this element is optional, exactly one of either <returnVoided> or <itemReturnResult> must appear in the return result. Thus, when this element is present, it must always be **true**.

Offline Return Result

Finally, Mr. Smith decides that he does, in fact, want to return the baseballs. When he returns to do so, the point-of-return is offline from Returns Management. The point-of-return makes it own decisions about the returnability of the items. For the sake of illustration, the point-of-return makes exactly the same decisions that Returns Management did previously, though there is no requirement for this.

> **Note:** The <itemreturnInfo> content is the same as in Example 5-1 and has been left out for brevity.

Example 7–7 Offline Return Result

```
<ReturnResult>
    <offlineDate>2006-05-16</offlineDate>
    <offlineRequest>
        <itemReturnInfo>
            ...
        </itemReturnInfo>
        <returnStoreID>04241</returnStoreID>
        <returnWorkstationID>123</returnWorkstationID>
        <employeeID>20051</employeeID>
        <customerInfo>
            <customerID>8885551212</customerID>
        </customerInfo>
        <positiveID>
            <number>12345678</number>
            <type>DriversLicense</type>
            <issuer>US_MN</issuer>
            <issued>2004-01-01</issued>
            <expiration>2007-01-01</expiration>
        </positiveID>
        <transactionType>Return</transactionType>
    </offlineRequest>
    <returnTransactionID>
        <storeID>04241</storeID>
        <workstationID>123</workstationID>
        <sequenceNumber>263</sequenceNumber>
        <businessDate>2006-10-25</businessDate>
    </returnTransactionID>
    <itemReturnResult>
        <itemIdentifier>
            <itemID>40020002</itemID>
        </itemIdentifier>
        <quantityReturned>10</quantityReturned>
        <finalResultCode>Authorized</finalResultCode>
        <overrideInfo>
            <managerID>20008</managerID>
            <overrideObtained>true</overrideObtained>
            <tenderOverride>false</tenderOverride>
        </overrideInfo>
        <originalTransactionID>
            <storeID>12345</storeID>
            <workstationID>124</workstationID>
            <sequenceNumber>2</sequenceNumber>
            <businessDate>2005-12-31</businessDate>
        </originalTransactionID>
        <returnTender>
            <tenderType>
```

```
                <type>CASH</type>
                <amount>40.00</amount>
            </tenderType>
        </returnTender>
    </itemReturnResult>
</ReturnResult>
```

The offline result is effectively a return result with two additional elements. The first element is the date of the offline return. The second element is a return request message encapsulated in the <offlineRequest> element.

<offlineDate>, optional, date
This element is the original date of the offline return. Though this element is optional in the XSD, it is required when processing an offline return.

<offlineRequest>, complex type
The <offlineRequest> element is an embedded return request. It has the exact same format as a normal return request message except that the root <ReturnRequest> element is replaced by the <offlineRequest> element.

> **Note:** The XSD specifies that the result message has either an <offlineRequest> element or a <returnTicketID> element, but not both.

<returnTransactionID>, optional, complex type
This element refers to the ARTS-compliant return transaction generated by the point-of-return. It has the same format as the <transactionID> element of the return request.

<itemReturnResult>, complex type
This element contains the detailed data about the items returned in this offline request. It has the same format as the original return result described above.

Implementation Decisions

The following are types of implementation decisions.

Asynchronous Versus Synchronous Communication

Synchronous communication involves a client sending a message to a server and then pausing while it waits for a response. Asynchronous communication enables a client to send a message to the server and then immediately resume operations. Synchronous communication is usually more straightforward than asynchronous communication, but increases binding between a client and a server and can degrade concurrency. Asynchronous communication has the opposite problems: it is usually more complicated, but encourages decoupling and throughput.

Additionally, synchronous communication is necessary when a client needs a time-sensitive response to a message.

Returns Management prefers to use asynchronous communication. This is mainly due to concurrency. When communication is asynchronous, it can be off-loaded onto a lightly loaded machine or scheduled to run at a time when there is relatively little system activity. Also, the communication can be more easily chained, for example, by inserting an arbitrary number of message forwarders between the client and server, or by inserting a message broadcaster. This enables greater flexibility in future system growth. However, asynchronous messaging is poorly suited for real time responses to messages.

Because asynchronous messaging provides greater latitude at installation and higher concurrency, the result message is implemented as an asynchronous call. Evaluation, however, has a strong requirement for a rapid response, for example, a customer is physically waiting at the point-of-return for a return approval. This evaluation is implemented as a synchronous call.

XML Versus JavaBean Messages

Extensible markup language (XML) is a text format that is language independent and human legible. It has wide support in a variety of programming languages and a robust description language, XML Schema Definition (XSD). Being a text format, XML is capable only of encoding data.

JavaBeans are Java language constructs that mainly encapsulate data in an object class. Being objects, however, JavaBeans can optionally contain behavior as well as data. Also because they are objects, both the originator and the receiver of a JavaBean must have access to compatible versions of the .class file.

JavaBeans are a well known Java idiom and have a great deal of support in the Java Development Kit (JDK). However, they are a Java-specific solution. Furthermore, XML is more accessible to a non-technical audience than either Java source or Java runtime debugging environments. Also, XML is the standard of Web Service communication. Therefore, the messages that Returns Management passes are XML based rather than JavaBean (or Java Object) based. The XML is eventually transformed into JavaBeans. This transformation to and from JavaBeans is facilitated by Java XML Binding (JAXB) code.

Web Service Versus Enterprise JavaBeans and Remote Method Invocation Call

Web Services are language neutral, similar to XML. Web Services also provide a well-defined publishing and discovery mechanism: Universal Description, Discovery and Integration (UDDI).

Note: All POS Suite Web services are deployed on an Axis2 1.5.4 framework. An Apache Rampart 1.5.1 security module is used for authentication.

Elements

The previous sections have discussed details of the XML messages. This section provides more information about important elements.

Return Request

In Table 7–2, the mark (x) indicates that the element is required and needs to have an appropriate value for that scenario. A value **true** or **false** indicates that this element should be set to this explicit value.

The **Element** column represents the various elements; the other columns represent the different scenarios. The XPath expressions clarify where the elements are in relationship to the entire message.

Note: In subsequent Positive ID messages, all original scenario elements should be sent.

Table 7–2 **Required Elements By Return Request**

		Receipted			
Element	**Non-receipted**	**No transaction data**	**Has data, no transaction found**	**Has data, transaction found**	**Positive ID**
/ReturnRequest/itemReturnInfo/itemTransactionInfo					
receipted	false	true	true	true	NA
transactionID	NA	NA	X	X	NA
found	false	false	false	true	NA
validAtPointOfReturn	false	false	X	true	NA
validationAmount	NA	NA	X	NA	NA
originalTender	X	X	X	X	NA
/ReturnRequest/itemReturninfo					
itemIdentifier	X	X	X	X	NA
returnReason	X	X	X	X	NA
quantity	X	X	X	X	NA
manuallyEntered	false	false	X	false	NA
/ReturnRequest					
returnStoreID	X	X	X	X	NA
returnWorkStationID	X	X	X	X	NA
employeeID	X	X	X	X	NA
customerInfo or moreCustomerInfo	X	X	X	X	NA
EncryptedPositiveID	NA	NA	NA	NA	X
transactionType	X	X	X	X	NA
returnTicketID	NA	NA	NA	NA	X

Return Response

In Table 7–3, the **Element** column refers to the elements in the return response. The remaining columns describe the various use cases. A mark (x) means that the point-of-return should be concerned with this data point when encountering it. A **true** or **false** value means that the point-of-return will examine this data point for this value to determine the use case. The XPath expressions help the reader orient themselves with the elements.

This is the minimum amount of data a point-of-return needs to implement. Additionally, the point-of-return must decide how to interpret the approval or denial of the <responseApproveDenyCode> element.

Keep in mind that some items might be approved while others are denied.

Table 7–3 **Required Elements By Return Response**

Element	Approval or Denial	Positive ID Required	CS Override
/ReturnResponse			
returnTicketID	X	X	X
responseApproveDenyCode	X	NA	X
availableCustomerServiceOverride	NA	NA	true
/ReturnResponse/itemReturnResponse			
itemIdentifier	X	X	X
approveDenyCode	X	NA	NA
refundTenders	X	NA	X
customerInfoRequired	NA	true	NA

Return Result

In Table 7–4, the **Element** column is the list of elements the point-of-return needs to send in the ReturnResult message. A mark (x) indicates that this element should be present when sending this type of response to Returns Management. A **true** or **false** value indicates that this element should be set explicitly to this value when sending this type of message. The XPath expressions help orient the reader.

Table 7–4 **Required Elements By Return Result Use Case**

Element	Standard Result	Offline Return	Voided Return
/ReturnResult			
returnticketID	X	NA	X
offlineDate	NA	X	NA
offlineRequest	NA	X	NA
returnTransactionID	X	X	X
returnVoided	NA	NA	true
/ReturnResult/itemReturnResult			
itemIdentifier	X	X	NA
quantityReturned	X	X	NA
finalResultCode	X	X	NA
overrideInfo[1]	X	X	NA
returnTender	X	X	NA

[1] The <overrideInfo> element should be included only if there was an override.

Web Service Interface

The Web service exposes two methods. Table 7–5 describes these methods, with their parameters.

Table 7–5 **Web Service Methods**

Method Name	Input	Output
evaluateReturnRequest	String (ReturnRequest)	String (ReturnResponse)
processFinalResult	String (ReturnResult)	None

Note that though the Web services expect to produce and consume XML, the XML is passed and returned as a simple string rather than a DOM object.

By default, the Web service will be accessed at:

`http://hostname:port/retwebsvc/services/ReturnsManager`

Where *hostname*:*port* is replaced with the host and port to which the Web service .ear file is deployed. The retwebsvc context can be modified by changing the context-root element of the WebModule_Returns_WebServices module in the `application.xml` of the deployed ear.

Relationship of Returns Management Data to ARTS Transaction Data

The Association for Retail Technology Standards (ARTS) is an international membership organization dedicated to reducing the costs of technology through standards. ARTS has four standards:

- The Standard Relational Data Model
- UnifiedPOS
- IXRetail
- Standard RFPs

For more information about ARTS, go to:

`http://www.nrf-arts.org/`

One of the design goals of Returns Management is to reduce its dependency on external systems. At the same time, customers will need traceability of Returns Management data back to original transaction data.

To account for this, the return request and return result messages sent to Returns Management contain the ARTS-compliant transaction IDs for the relevant transactions. Therefore, when a return request is made, Returns Management is told of the transaction ID, if any, of the original sale. When a return result is sent, Returns Management is told of the transaction ID of the return transaction. This ID is stored with the return ticket.

8

Oracle Retail Point-of-Service to Oracle Retail Store Inventory Management Architecture

The Oracle Retail Point-of-Service-to-Store Inventory Management integration is intended to provide integration for the Point-of-Service application to interact with Store Inventory Management for inventory information. The following features are supported for integration with an inventory management system:

- Inventory Inquiry: This feature is provided to enable Point-of-Service to check the item inventory in Home Store, Buddy Store, Specific Store and Transfer zone. The Item Inventory feature is available to Point-of-Service client only when the Point-of-Service client is in the Online mode.
- Item Basket: This feature is provided for line busting using the Store Inventory Management handheld. The items in a customer basket are scanned using the Store Inventory Management handheld and staged in the Store Inventory Management database. Point-Of-Service can then look up the basket details and add the line items to the sell item screen.
- Serial Number Validation and Update: Point-Of-Service supports serialized items. The operator is prompted to enter/scan the serial number of the serialized item on the Point-of-Service client. The serial number that is entered is then validated by interfacing with Store Inventory Management. Once the transaction is tendered, the serialized items along with the captured serial number will be sent to Store Inventory Management for updating the status of the particular serial number.
- Inventory Reservation: Point-of-Service interfaces with Store Inventory Management to send the order transactions so that the items can be marked as reserved in Store Inventory Management. Also, once the items are picked up or delivered to the customer, the status needs to be updated in Store Inventory Management.
- Real Time Inventory Status Update: This interface sends Point-of-Service transactions to Store Inventory Management to update the inventory status based on the transactions.

The following steps outline the Point-of-Service-to-Store Inventory Management integration approach:

1. Expose the inventory features from Store Inventory Management in the form of Web service.
2. Provide pluggable inventory Web service interface to integrate Point-of-Service-to-Store Inventory Management.

3. Point-of-Service client interacts with Point-of-Service server over RMI as in the existing Point-of-Service architecture. Point-of-Service server interacts with inventory Web service interface to interact with Store Inventory Management.
4. Point-of-Service uses the connector framework to achieve a pluggable and extendable integration with Store Inventory Management.

The Point-of-Service to Store Inventory Management integration system is broken into five main sub-systems:

- Point-of-Service Client
- Point-of-Service Server
- Point-of-Service COMMEXT (Communication Extension Module)
- Store Inventory Management Server
- Store Inventory Management DB

Point-of-Service Client

The various functionalities are incorporated in Point-of-Service client by having new tours and new components, namely the ConnectorManager for interaction with the ConnectorTechnician.

Point-of-Service Server

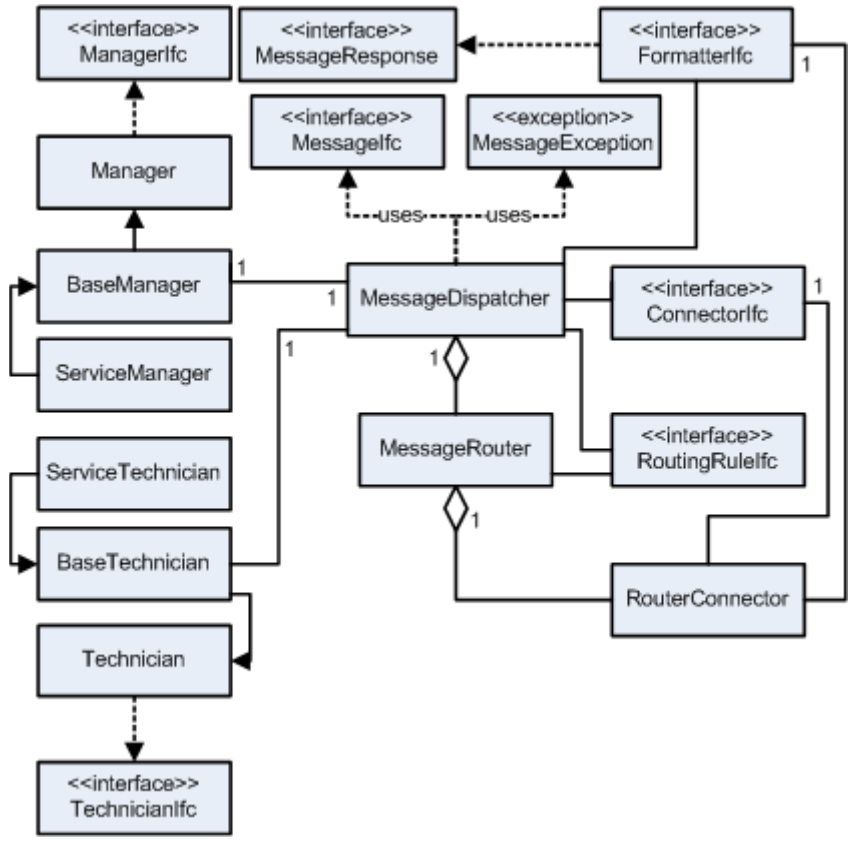
The Point-of-Service server contains the connector framework which embeds the integration details. The connector framework is exposed through the ConnectorManager and RetailTransactionTechnician. The connector framework consists of pluggable Formatters (request-response formatting) and Connectors (ORSIMWebServiceConnector) to abstract the connection-specific logic.

The ORSIMWebServiceConnector is in the Point-of-Service server. PSITechnician interacts with PSIInventoryWS_Stub to call InventoryWS over intranet using HTTP/SOAP protocol.

Point-of-Service COMMEXT (Communication Extension Module)

The COMMEXT (Communication Extension Module) is an out-of-the-box integration framework. It provides a very extendable approach to the integrations both online and offline. The Point-of-Service COMMEXT model is as shown in Figure 8–1. The separation of concerns between data structure manipulation or transformation, and handling connectivity to a service is separated between the two components–the formatter and the connector.

Figure 8–1 Point-of-Service Connector Framework Model



The MessageDispatcher is the core of the communication framework. Its primary function is to dispatch messages to mapped routers. In addition, MessageDispatcher performs administrative and control operations on the associated connectors. When invoked, the MessageDispatcher delegates the message handling to a specific MessageRouter.

The MessageRouter coordinates the processing of a message using the associated routing rule and the RouterConnectors.

A RouterConnector provides an association between a message type, connector, and formatter. This decouples the formatting of the message from the chosen connector.

ConnectorIfc handles the communication between the application and the external service. It is responsible for locating the service, establishing a connection, and interacting with the service using appropriate protocols.

FormatterIfc translates the raw data from the message into the format expected by the external service. It also translates the response from the remote service into the format expected by the application.

Once a message has been sent with a request type to the MessageDispatcher it will get the instance of MessageRouter that is configured for that request type from the instantiated list. The processing is then delegated to the MessageRouter. The MessageRouter will route the request message to the list of connectors that are configured for that request. There can be multiple connectors that can be defined to process the same request message.

The connector framework provides all the building blocks to realize any integration requirement with a combination of connectors, formatters, ChainedConnectors, RoutingRules and JMX notifications. The XML configuration ties up the various blocks to implement any integration requirement.

Store Inventory Management Server
Inventory Web service component deployed in Store Inventory Management server provides the entry point into the application for the various functionalities.

Store Inventory Management DB
Store Inventory Management inventory database.

Item Disposition

The retailer can map the SIM inventory adjustment reason codes with the Point-of-Service reason codes and send it to SIM in the Web service call.

SIM uses these reason codes to identify the item disposition against the reason code and updates the inventory buckets appropriately. SIM processes the Web service call and increments the SOH, performing the inventory adjustment based on the disposition.

The following item dispositions are the valid mapped dispositions:

- Available to Sell (ATS) to TRBL -- This disposition moves the inventory from Available to Unavailable. For the retailer, this means the stock is taken in and made unavailable to sell.
- ATS to Distributed (DIST) -- This disposition moves the inventory from Available to Out of inventory. End result the SOH is incremented and then again decremented. For the store person, this means the return is accepted and the item which was returned is not in a condition to keep it back on the rack and it is destroyed.

Error Handling

Error handling is limited to logging errors during the inventory lookup. The exceptions such as IOException and invalidItem that occur during WSService communication are re-thrown as WSException, as well as logged for error tracking and resolution.

Logging

Point-of-Service to Store Inventory Management uses Log4J for logging. The following logging levels can be used:

- Info: For logging information messages.
- Debug: For logging all the debug messages.
- Error: For logging application errors.

The logging level can be configured with log4J.xml. See Configuring Logging in Point-of-Service for more information.

9

Changing and Configuring Currencies

This chapter describes how to change currencies as well as configure new currencies.

Changing Currency

To switch to another base and alternate currency, perform the following steps:

1. Set the base currency flag in the primary currency of the currency table. For example, if EUR is the base currency:

   ```
   update co_cny set FL_CNY_BASE='1' where DE_CNY='EUR'
   ```

2. Remove the base currency flag from any other currencies in that table. For example:

   ```
   update co_cny set FL_CNY_BASE = '0' where DE_CNY <> 'EUR'
   ```

3. Enforce ordering so that the primary currency is first and the alternate currency is second for the AI_CNY_PRI column in the currency table. Other rows should be ordered, but the specific order is not important. For example, if EUR is base currency and GBP is the alternate:

   ```
   update co_cny set AI_CNY_PRI=0 where DE_CNY='EUR'
   update co_cny set AI_CNY_PRI=1 where DE_CNY='GBP'
   update co_cny set AI_CNY_PRI=2 where DE_CNY='USD'
   update co_cny set AI_CNY_PRI=3 where DE_CNY='CAD'
   update co_cny set AI_CNY_PRI=4 where DE_CNY='MXN'
   update co_cny set AI_CNY_PRI=5 where DE_CNY='JPY'
   ```

4. Add store safe tenders supported for the new base and alternate currency. For example, if EUR is the new base currency, add money order tender support for EUR:

   ```
   insert into le_tnd_str_sf
   (ID_RPSTY_TND, TY_TND, TY_SB_TND, LU_CNY_ISSG_CY, TS_CRT_RCRD, TS_MDF_RCRD, ID_
   CNY_ICD )
   VALUES ('1','MNYO', ' ', 'EU', CURRENT_TIMESTAMP, CURRENT_TIMESTAMP, 5);
   ```

   Remove store safe tenders no longer supported for the old base/alternate currency. For example, if USD if the old base currency, remove money order tender support for USD:

   ```
   delete from le_tnd_str_sf where LU_CNY_ISSG_CY = 'US' and TY_TND = 'MNYO';
   ```

5. Add exchange rate records for alternate and base currencies into the CO_RT_EXC table based on the new base currency. Delete all exchange rate records based on any previous base currency.

There are some application parameters that must be changed as well:

- Tender Group:
  - `CashAccepted`: For example, if EUR is base and GBP is alternate, make sure that the `CashAccepted` parameter is changed so that EUR and GBP are selected.
  - `TravelersChecksAccepted`: For EUR as base and GBP as alternate, the values for the `TravelersChecksAccepted` parameter should be EURCHK and GBPCHK.
  - `ChecksAccepted`: For EUR as base and GBP as alternate, the values for the `ChecksAccepted` parameter should be EURCHK and GBPCHK.
  - `GiftCertificateAccepted`: Change the values to reflect all the currencies accepted (base and alternate). For example the values may be EUR and GBP, or EUR, GBP and USD.
  - `StoreCreditAccepted`: Change the values to reflect all the currencies accepted (base and alternate). For example the values may be EUR and GBP, or EUR, GBP and USD.
- Reconciliation Group:
  - `TendersToCountAtTillReconcile`: For EUR as base and GBP as alternate, the values for the `TendersToCountAtTillReconcile` parameter should be:
    * Cash
    * Check
    * Credit
    * Debit
    * TravelCheck
    * GiftCert
    * Coupon
    * GiftCard
    * StoreCredit
    * MallCert
    * PurchaseOrder
    * MoneyOrder
    * GBPCash
    * GBPTravelCheck
    * GBPCheck
    * GBPGiftCert
    * GBPStoreCredit

Configuring a New Base Currency

Throughout this section, Krona is used as the example new base currency that is being configured. The Krona currency code is SEK, and the issuing country code is SE.

Currency SQL Configuration

The following SQL configurations for Currency are available.

Currency Table CO_CNY

A new record describing the new currency information such as its currency code, issuing country code and so forth, must be inserted into this table.

In the base currency flag column **FL_CNY_BASE**, the new currency must be set to **1** indicating that it is the base. The flag for other currencies must be set to **0**, indicating that they are alternate currencies.

> **Note:** Point-of-Service supports base-plus-one alternate currency. The priority column AI_CNY_PRI must be set to 0 for the new base currency. It must be set to 1 for the supported alternate currency. For other alternate currencies, they must be ordered and greater than 1, but the specific order isn't important.

Example 9–1 Add Krona as Base to Currency Table CO_CNY

```
INSERT INTO CO_CNY
(ID_CNY_ICD, LU_CNY_ISSG_CY, CD_CNY_ISO, DE_CNY, DE_CNY_ISSG_NAT, FL_CNY_BASE, QU_
CNY_SCLE, AI_CNY_PRI)
VALUES (7,'SE', 'SEK', 'SEK', 'Sweden', '1', 2, 0);

UPDATE CO_CNY
SET FL_CNY_BASE = '0'
WHERE CD_CNY_ISO <> 'SEK';

UPDATE CO_CNY
SET AI_CNY_PRI = AI_CNY_PRI + 1
WHERE CD_CNY_ISO <> 'SEK';
```

Currency Denomination Table CO_CNY_DNM and I8 table CO_CNY_DNM_I8

Denominations for the new base currency must be added to the CO_CNY_DNM and CO_CNY_DNM_I8 table. For example:

Example 9–2 Add Krona Denominations to Denomination Table CO_CNY_DNM

```
INSERT INTO CO_CNY_DNM
(ID_CNY_ICD, ID_CNY_DNM, NM_DNM, VL_DNM, CD_DNM_DPLY_PRI)
VALUES (7, 1, 'SE_50Ores', '0.50', 1);

INSERT INTO CO_CNY_DNM
(ID_CNY_ICD, ID_CNY_DNM, NM_DNM, VL_DNM, CD_DNM_DPLY_PRI)
VALUES (7, 2, 'SE_1Kronas', '1.00', 2);

INSERT INTO CO_CNY_DNM
(ID_CNY_ICD, ID_CNY_DNM, NM_DNM, VL_DNM, CD_DNM_DPLY_PRI)
VALUES (7, 3, 'SE_5Kronas', '5.00', 3);

INSERT INTO CO_CNY_DNM
```

```
(ID_CNY_ICD, ID_CNY_DNM, NM_DNM, VL_DNM, CD_DNM_DPLY_PRI)
VALUES (7, 4, 'SE_10Kronas', '10.00', 4);

INSERT INTO CO_CNY_DNM
(ID_CNY_ICD, ID_CNY_DNM, NM_DNM, VL_DNM, CD_DNM_DPLY_PRI)
VALUES (7, 5, 'SE_20Kronas', '20.00', 5);

INSERT INTO CO_CNY_DNM
(ID_CNY_ICD, ID_CNY_DNM, NM_DNM, VL_DNM, CD_DNM_DPLY_PRI)
VALUES (7, 6, 'SE_50Kronas', '50.00', 6);

INSERT INTO CO_CNY_DNM
(ID_CNY_ICD, ID_CNY_DNM, NM_DNM, VL_DNM, CD_DNM_DPLY_PRI)
VALUES (7, 7, 'SE_100Kronas', '100.00', 7);

INSERT INTO CO_CNY_DNM
(ID_CNY_ICD, ID_CNY_DNM, NM_DNM, VL_DNM, CD_DNM_DPLY_PRI)
VALUES (7, 8, 'SE_1000Kronas', '1000.00', 8);
```

Example 9–3 Add Krona Denominations to I8 Table CO_CNY_DNM_I8

```
INSERT INTO CO_CNY_DNM_I8
(ID_CNY_ICD, ID_CNY_DNM, LCL, NM_DNM)
VALUES (7, 2,'en','1 Kronas');

INSERT INTO CO_CNY_DNM_I8
(ID_CNY_ICD, ID_CNY_DNM, LCL, NM_DNM)
VALUES (7, 2,'fr','1 couronne');
```

Note: For each denomination record in the CON_CNY_DNM table, there are I8 records in CO_CNY_DNM_I8 table, one for each supported language.

Exchange Rate Table CO_RT_EXC

Add exchange rate records for alternate and base currencies into the CO_RT_EXC table based on the new base currency. Delete all exchange rate records based on any previous base currency. For example:

Example 9–4 Add Alternate Currency Exchange Rates to Krona

```
-- Delete all the existing records
Delete from CO_RT_EXC;

INSERT INTO CO_RT_EXC
(LL_CNY_EXC, DC_RT_EXC_EF, DC_RT_EXC_EP, ID_CNY_ICD, MO_RT_TO_BUY, MO_RT_TO_SL,
MO_FE_SV_EXC)
VALUES(0.00, TO_DATE('1990-01-01', 'YYYY-MM-DD'), TO_DATE('2099-12-31',
'YYYY-MM-DD'), 1, 6.3337, 6.3362, 0.00);

INSERT INTO CO_RT_EXC
(LL_CNY_EXC, DC_RT_EXC_EF, DC_RT_EXC_EP, ID_CNY_ICD, MO_RT_TO_BUY, MO_RT_TO_SL,
MO_FE_SV_EXC)
VALUES(0.00, TO_DATE('1990-01-01', 'YYYY-MM-DD'), TO_DATE('2099-12-31',
'YYYY-MM-DD'), 2, 6.2849, 6.2898, 0.00);

INSERT INTO CO_RT_EXC
(LL_CNY_EXC, DC_RT_EXC_EF, DC_RT_EXC_EP, ID_CNY_ICD, MO_RT_TO_BUY, MO_RT_TO_SL,
MO_FE_SV_EXC)
```

```
VALUES(0.00, TO_DATE('1990-01-01', 'YYYY-MM-DD'), TO_DATE('2099-12-31',
'YYYY-MM-DD'), 3, 0.5799, 0.5816, 0.00);

INSERT INTO CO_RT_EXC
(LL_CNY_EXC, DC_RT_EXC_EF, DC_RT_EXC_EP, ID_CNY_ICD, MO_RT_TO_BUY, MO_RT_TO_SL,
MO_FE_SV_EXC)
VALUES(0.00, TO_DATE('1990-01-01', 'YYYY-MM-DD'), TO_DATE('2099-12-31',
'YYYY-MM-DD'), 4, 12.434, 12.441, 0.00);

INSERT INTO CO_RT_EXC
(LL_CNY_EXC, DC_RT_EXC_EF, DC_RT_EXC_EP, ID_CNY_ICD, MO_RT_TO_BUY, MO_RT_TO_SL,
MO_FE_SV_EXC)
VALUES(0.00, TO_DATE('1990-01-01', 'YYYY-MM-DD'), TO_DATE('2099-12-31',
'YYYY-MM-DD'), 5, 9.3739, 9.3796, 0.00);

INSERT INTO CO_RT_EXC
(LL_CNY_EXC, DC_RT_EXC_EF, DC_RT_EXC_EP, ID_CNY_ICD, MO_RT_TO_BUY, MO_RT_TO_SL,
MO_FE_SV_EXC)
VALUES(0.00, TO_DATE('1990-01-01', 'YYYY-MM-DD'), TO_DATE('2099-12-31',
'YYYY-MM-DD'), 6, 0.05782, 0.05786, 0.00);

INSERT INTO CO_RT_EXC
(LL_CNY_EXC, DC_RT_EXC_EF, DC_RT_EXC_EP, ID_CNY_ICD, MO_RT_TO_BUY, MO_RT_TO_SL,
MO_FE_SV_EXC)
VALUES(0.00, TO_DATE('1990-01-01', 'YYYY-MM-DD'), TO_DATE('2099-12-31',
'YYYY-MM-DD'), 7, 1.0, 1.0, 0.00);
```

Store Safe Tender Table LE_TND_STR_SF

Add the store safe tenders supported for the new base currency. For example:

Example 9–5 Add Store Safe Tenders for Krona

```
INSERT INTO LE_TND_STR_SF
    ( ID_RPSTY_TND, TY_TND, TY_SB_TND, LU_CNY_ISSG_CY, TS_CRT_RCRD, TS_MDF_RCRD,
ID_CNY_ICD )
    VALUES('1','CASH', ' ', 'SE', CURRENT_TIMESTAMP, CURRENT_TIMESTAMP, 7);
INSERT INTO LE_TND_STR_SF
    ( ID_RPSTY_TND, TY_TND, TY_SB_TND, LU_CNY_ISSG_CY, TS_CRT_RCRD, TS_MDF_RCRD,
ID_CNY_ICD )
    VALUES('1','CHCK', ' ', 'SE', CURRENT_TIMESTAMP, CURRENT_TIMESTAMP, 7);
INSERT INTO LE_TND_STR_SF
    ( ID_RPSTY_TND, TY_TND, TY_SB_TND, LU_CNY_ISSG_CY, TS_CRT_RCRD, TS_MDF_RCRD,
ID_CNY_ICD )
    VALUES('1','TRAV', ' ', 'SE', CURRENT_TIMESTAMP, CURRENT_TIMESTAMP, 7);

-- MoneyOrderSafeTender

INSERT INTO LE_TND_STR_SF
(ID_RPSTY_TND, TY_TND, TY_SB_TND, LU_CNY_ISSG_CY, TS_CRT_RCRD, TS_MDF_RCRD, ID_
CNY_ICD )
VALUES ('1','MNYO', ' ', 'SE', CURRENT_TIMESTAMP, CURRENT_TIMESTAMP, 7);
```

Money Order Tenders are only accepted for base currency, therefore before inserting records for the new base currency, delete any money order tenders for the other currencies:

```
DELETE * from LE_TND_STR_SF where ty_tnd='MNYO'
```

Parameter Configuration

The following tender parameters must be enhanced to include the new base currency:

- StoreCreditsAccepted
- ChecksAccepted
- CashAccepted
- GiftCertificatesAccepted
- TravelersChecksAccepted

The reconciliation parameter **TendersToCountAtTillReconcile** parameter must include all the tenders to count for both base and alternate currencies during till reconciliation. For example:

Example 9–6 Parameters to support Krona as the base and USD as the alternate currency

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE SOURCE PUBLIC "SOURCE"
"classpath://com/extendyourstore/foundation/tour/dtd/paramsourcescript.dtd">
<SOURCE name="register">
<GROUP hidden="N" name="Tender">
<PARAMETER final="N" hidden="N" name="StoreCreditsAccepted" type="LIST">
<VALIDATOR class="EnumeratedListValidator"
package="oracle.retail.stores.foundation.manager.parameter">
<PROPERTY propname="member" propvalue="None"/>
<PROPERTY propname="member" propvalue="USD"/>
<PROPERTY PROPNAME="MEMBER" PROPVALUE="SEK"/>
<PROPERTY PROPNAME="MEMBER" PROPVALUE="EUR"/>
</VALIDATOR>
<VALUE value="SEK"/>
<VALUE value="USD"/>
<VALUE value="EUR"/>
</PARAMETER>
<PARAMETER final="N" hidden="N" name="ChecksAccepted" type="LIST">
<VALIDATOR class="EnumeratedListValidator"
package="oracle.retail.stores.foundation.manager.parameter">
<PROPERTY propname="member" propvalue="None"/>
<PROPERTY propname="member" propvalue="USDCHK"/>
<PROPERTY propname="member" propvalue="SEKCHK"/>
<PROPERTY propname="member" propvalue="EURCHK"/>
</VALIDATOR>
<VALUE value="SEKCHK"/>
<VALUE value="USDCHK"/>
</PARAMETER>
<PARAMETER final="N" hidden="N" name="CashAccepted" type="LIST">
<VALIDATOR class="EnumeratedListValidator"
package="oracle.retail.stores.foundation.manager.parameter">
<PROPERTY propname="member" propvalue="None"/>
<PROPERTY propname="member" propvalue="USD"/>
<PROPERTY propname="member" propvalue="SEK"/>
<PROPERTY propname="member" propvalue="EUR"/>
</VALIDATOR>
<VALUE value="SEK"/>
<VALUE value="USD"/>
</PARAMETER>
<PARAMETER final="N" hidden="N" name="GiftCertificatesAccepted" type="LIST">
<VALIDATOR class="EnumeratedListValidator"
package="oracle.retail.stores.foundation.manager.parameter">
```

```
<PROPERTY propname="member" propvalue="None"/>
<PROPERTY propname="member" propvalue="USD"/>
<PROPERTY propname="member" propvalue="SEK"/>
<PROPERTY propname="member" propvalue="EUR"/>
</VALIDATOR>
<VALUE value="SEK"/>
</PARAMETER>
<PARAMETER final="N" hidden="N" name="TravelersChecksAccepted" type="LIST">
<VALIDATOR class="EnumeratedListValidator"
package="oracle.retail.stores.foundation.manager.parameter">
<PROPERTY propname="member" propvalue="None"/>
<PROPERTY propname="member" propvalue="USDCHK"/>
<PROPERTY propname="member" propvalue="SEKCHK"/>
<PROPERTY propname="member" propvalue="EURCHK"/>
</VALIDATOR>
<VALUE value="SEKCHK"/>
<VALUE value="USDCHK"/>
Configuring a New Base Currency
Appendix: Changing and Configuring a New Base Currency D-7
</PARAMETER>
</GROUP>
<GROUP hidden="N" name="Reconciliation">
<PARAMETER final="N" hidden="N" name="TendersToCountAtTillReconcile" type="LIST">
<VALIDATOR class="EnumeratedListValidator"
package="oracle.retail.stores.foundation.manager.parameter">
<PROPERTY propname="member" propvalue="Cash"/>
<PROPERTY propname="member" propvalue="Check"/>
<PROPERTY propname="member" propvalue="ECheck"/>
<PROPERTY propname="member" propvalue="Credit"/>
<PROPERTY propname="member" propvalue="Debit"/>
<PROPERTY propname="member" propvalue="TravelCheck"/>
<PROPERTY propname="member" propvalue="GiftCert"/>
<PROPERTY propname="member" propvalue="Coupon"/>
<PROPERTY propname="member" propvalue="GiftCard"/>
<PROPERTY propname="member" propvalue="StoreCredit"/>
<PROPERTY propname="member" propvalue="MallCert"/>
<PROPERTY propname="member" propvalue="PurchaseOrder"/>
<PROPERTY propname="member" propvalue="MoneyOrder"/>
<PROPERTY propname="member" propvalue="USDCash"/>
<PROPERTY propname="member" propvalue="USDTravelCheck"/>
<PROPERTY propname="member" propvalue="USDCheck"/>
<PROPERTY propname="member" propvalue="USDGiftCert"/>
<PROPERTY propname="member" propvalue="USDStoreCredit"/>
</VALIDATOR>
<VALUE value="Cash"/>
<VALUE value="Check"/>
<VALUE value="ECheck"/>
<VALUE value="Credit"/>
<VALUE value="Debit"/>
<VALUE value="TravelCheck"/>
<VALUE value="GiftCert"/>
<VALUE value="Coupon"/>
<VALUE value="GiftCard"/>
<VALUE value="StoreCredit"/>
<VALUE value="MallCert"/>
<VALUE value="PurchaseOrder"/>
<VALUE value="MoneyOrder"/>
<VALUE value="USDCash"/>
<VALUE value="USDTravelCheck"/>
<VALUE value="USDCheck"/>
```

```
<VALUE value="USDGiftCert"/>
<VALUE value="USDStoreCredit"/>
</PARAMETER>
</GROUP>
</SOURCE>
```

Resource Bundle Configuration

New resource bundle keys that describe the new currency, including its issuing country, must be added to the following Point-of-Service resource bundles:

- commonText
- ejournalText
- tillText
- dailyOperationsText
- parameterText

Example 9–7 New commonText Resource Bundle Keys

```
#
# Supported Nationalities
Common.SE_Nationality=Swedish

#
# Supported Currencies
Common.SEK=Swedish Krona

#
# Supported Checks
Common.SEKCHK=Swedish Krona

#
#  Tender Types
#
Common.SEKCash=SEK Cash
Common.SEKCheck=SEK Check
Common.SEKTravCheck=SEK Trav. Check
```

Example 9–8 New ejournalText Resource Bundle Keys

```
JournalEntry.SEK=SEK
```

Example 9–9 tillText Resource Bundle Keys

```
SelectTenderSpec.SelectSEK=SEK

Add example for dailyOperations Resource Bundle Keys:

FinancialTotalsSummaryEntrySpec.CURRCODE_SE=SEK

Add example for parameterText Resource Bundle Keys:
Common.SEKCash=SEK Cash
Common.SEK TravelCheck=SEK  Traveler's Check
Common.SEK Check=SEK  Check
Common.SEK GiftCert=SEK  Gift Certificate
Common.SEK StoreCredit=SEK  Store Credit
Common.SEKGiftCard=SEK Gift Card
```

10

Configuring Multiple Printers for Labels and Tags

Oracle Retail Labels and Tags supports multiple printers that can be used for printing labels and tags. Users can select a printer from a list of printers on the Add Batch and Batch Detail screens.

For more information, see the *Oracle Retail Back Office Installation Guide*.

11

Returns Authorization

The integration allows for Point-of-Service to collect positive ID during the return transactions, to form and send the Return Request messages to Returns Management, to interpret and present the Returns Management Return Response messages, and to form and send Final Result messages to Returns Management right before the return transaction is completed. Point-of-Service has support/flow for all Returns Management Return Response types. Point-of-Service has support/flow for accepting and managing Returns Management recommended tenders.

Returns Management provides the ability to deliver an accept/deny response for attempted refunds on line items of return transactions as well as non-receipted return attempts through standard XML messages. The retailer can configure enterprise-wide, down to store- and item-specific, receipted and non-receipted policies that are applied to line items on transactions occurring at a point of sale or point of return. The policy definition as well as accept/deny logic is contained within the enterprise and therefore is abstracted from the point of sale or return such that Returns Management can work with any point of sale or return application, including Web or phone order systems. Returns Management provides the ability to count instances of behavior for customers and cashiers based on negativity activity and deny returns based on frequent suspicious activity. Included are inquiry screens to research an attempted refund or a particular score and its history.

Exception Flow

Communication with Returns Management is available only when Point-of-Service server is in Online mode. If Point-of-Service server goes offline at any time during authorization or sending Final Result, the authorization request and final result information will be saved in Point-of-Service as offline return information, the message in E-Journal is logged and the offline return information will be sent to Returns Management when it is available.

Error Handling

Error handling is limited to logging errors during the return authorization. The exceptions such as IOException and invalidItem that occur during WSService communication are re-thrown as WSException, as well as logged for error tracking and resolution.

Logging

RPI uses Log4J for Logging. The following logging levels can be used:

- **Info**: For logging information messages.
- **Debug**: For logging all the debug messages.
- **Error**: For logging application errors.
- **Warn**: For logging warning messages.

The logging level can be configured with log4J.xml.

12

Bill Pay

The bill pay feature in Point-of-Service enables retailers to accept bill payments from their customers and interface with their billing system to record the payments. This solution is primarily intended for telecom service providers who run their outlet stores primarily in developing markets.

Bill Pay provides an ability for a cashier at the store to accept bill payments and provides an integration of Point-of-Service with different billing systems, such as Oracle Billing and Revenue Management (BRM), Amdoc and so on.

Bill Pay provides the retailer with the following capabilities:

- Bill Search and Pay: The operator can scan the bill number to get the bill information and options to pay the bill using different tender types. The operator can also look up the bill details on a third-party billing system by providing customer information.
- Offline Bill Pay: When the third-party billing system is offline, Point-of-Service can take the payment by capturing the minimum information required for that bill payment and later sending this detail to the billing system when the system is online.
- Integration Framework: Enables the service implementers (SI) to integrate Point-of-Service with different third-party billing systems.

13

Authorized Payment Foundation

Authorized Payment Foundation Overview

The Authorized Payment Foundation (APF) provides a well-defined interface to third-party authorization services and the internal training mode's simulated authorization service by taking advantage of the established Manager/Technician framework and the new Communication Extension (COMMEXT) framework for integrations.

The Point-of-Service client tour sites and aisles populate APF request objects and pass the objects to the PaymentManager. The PaymentManager sends the request to the appropriate connectors and formatters using calls to the Communication Extension (COMMEXT) framework.

The base implementation of APF includes COMMEXT configurations for ACI PinComm and Servebase PC-EFT authorization services. Base implementation also includes a simulated authorization service for training mode.

APF Goals

The APF removes the handling of sensitive card account numbers from Point-of-Service, removes the direct integration between Point-of-Service and the CPOI device, and improves and isolates the interface between Point-of-Service and the authorization service.

Sensitive credit and debit card account numbers are not handled or persisted by Point-of-Service. The authorization service provides an account number token which is persisted and used in a variety of situations. The masked card number continues to be persisted and is also used.

Point-of-Service Client Flow Overview

Each tender is authorized as it is tendered. When a transaction is canceled, or when the tender option screen is left before the tendering is complete, all authorized tenders are reversed.

All CPOI interaction is performed using the authorization service. The granularity of CPOI control is defined by the third-party authorization service.

Implementing a New Authorization Service

If the new authorization service requires or returns information different from the information defined in the base APF request or response classes, changes might be required for:

- The APF request/response classes
- Intermediate Point-of-Service client authorization classses
- Database tables
- Data persistence classes/SQL

If the new authorization service supports additional features, then Point-of-Service client tours can be extended or modified. For more information see the *Oracle Retail POS Suite Implementation Guide – Volume 2, Extension Solutions*.

If the base APF request/response classes are adequate, then you might need to implement new COMMEXT connectors and formatters and modify the COMMEXT configuration only.

APF Request/Response Modifications

The APF classes are described in detail in APF Request Types. These classes might need to be modified to accommodate the new authorization service.

Database Modifications

The authorizable tender tables might require modification if information different from the authorization service response must be persisted.

Values returned from PinComm for credit or debit authorizations are stored in the TR_LTM_CRDB_CRD_TN table. Additional information for PC-EFT is stored in the TR_LTM_CRD_ICC table.

Point-of-Service Client Tour Modifications

If the new authorization service does not require additional information collection during the Point-of-Service client tour, then tour modifications might not be required.

The authorization service can support different features or require different information. If so, Point-of-Service client tours must be modified to collect the additional information and set the values in the APF request objects. For more information, see the *Oracle Retail POS Suite Implementation Guide – Volume 2, Extension Solutions*.

Many features in the base product are supported by PinComm but not supported by PC-EFT (such as House Accounts, Signature Capture, Gift Cards, Scrolling Receipts and Swipe-Ahead). These features are controlled by properties such as SignatureCaptureEnabled, ReturnByAccountNumberToken and POSGFCardTenderEntryRequired located in `<source_directory>\applications\pos\deploy\shared\config\application.properties`, and parameters such as GiftCardsAccepted and HouseAccountPayment located in `<source_directory>\applications\pos\deploy\shared\config\parameter\application\application.xml`. Some features, such as Scrolling Receipts, are enabled or disabled using COMMEXT filters. In many cases, tour modification can be avoided by changing these configurations.

COMMEXT Connectors/Formatters Implementation

New COMMEXT connectors and formatters must be implemented to enable the APF requests and responses to communicate with the requests and responses used by the new authorization service, and to send requests to the new service.

Connectors can also be used to handle more than just sending requests to the auth service. See PinComm Connectors for more information.

The PinComm connector calls an API to send requests to the authorization service. The PinComm implementation has unique formatters for each type of request that is sent to PinComm. The PinComm implementation also uses several connectors to handle special cases.

The PC-EFT connector opens a socket, through which the connector sends its request. The PC-EFT implementation uses a single formatter to format all types of requests sent to its authorization service.

Note: The implementer must determine which design to use for the new authorization service.

COMMEXT Configuration Modifications

The COMMEXT configuration file must be modified to use the new connectors and formatters.

The implementer must be familiar with COMMEXT configuration. For more information, see the *Oracle Retail POS Suite Implementation Guide – Volume 2, Extension Solutions*.

The following files contain the PinComm and PC-EFT configurations:

- PaymentManager.xml
- PaymentTechnician.xml

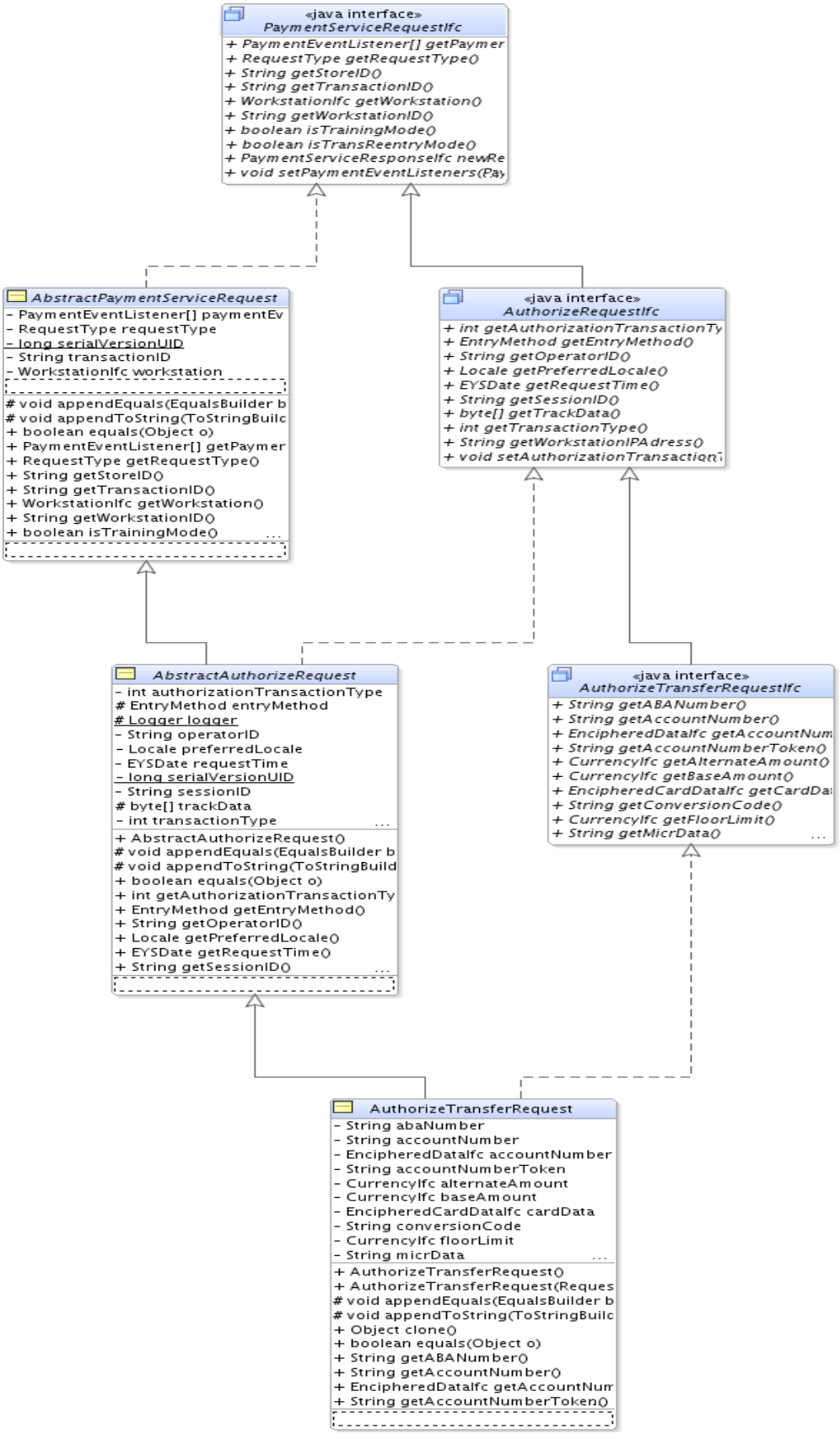
APF Request Types

APF request classes are object oriented and are organized to accommodate the different information required for the variety of requests and responses. Note that not all types are supported by the PC-EFT authorization service.

APF Authorize Payment (Transfer) Request Classes

This request is used to authorize payments.

Figure 13–1 AuthorizeTransferRequest Class



APF Reversal Request Classes

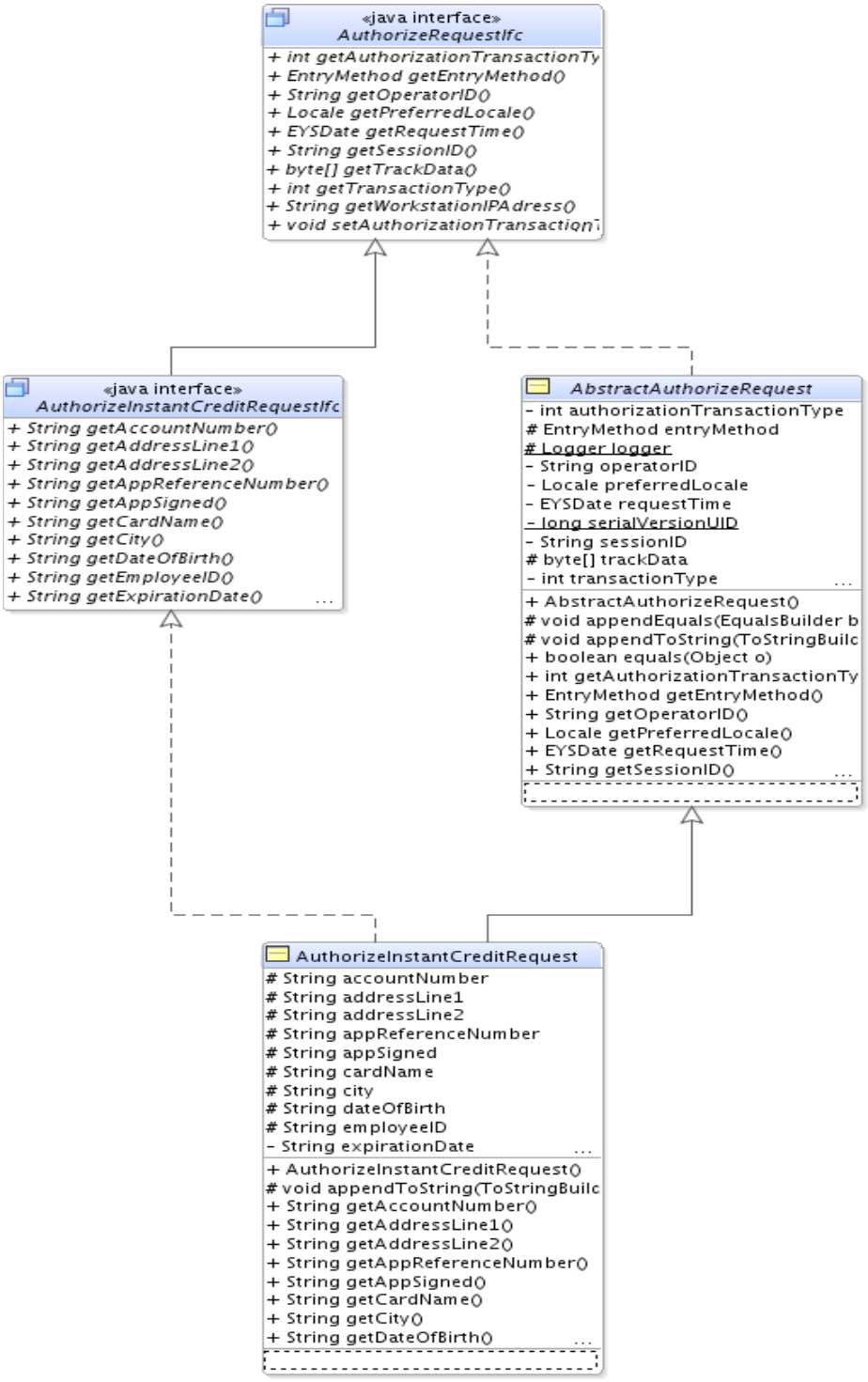
ReversalRequest inherits from some of the classes used by the authorize payment request. Those classes are excluded here to simplify the diagram. See Figure 13–1, "AuthorizeTransferRequest Class" for more detail. This request reverses previous authorizations. Reversals occur when a sale is canceled before the tendering is completed, and when the operator leaves the tendering screen before tendering is complete.

Figure 13–2 ReversalRequest Class

APF Instant Credit Request Classes

AuthorizeInstantCreditRequest inherits from some of the classes used by the authorize sale request. Those classes are excluded here to simplify the diagram. See Figure 13–1, "AuthorizeTransferRequest Class" for more detail. This request is used to apply for instant credit (house account) approval.

Figure 13–3 AuthorizeInstantCreditRequest Class



APF Call Referral Request Classes

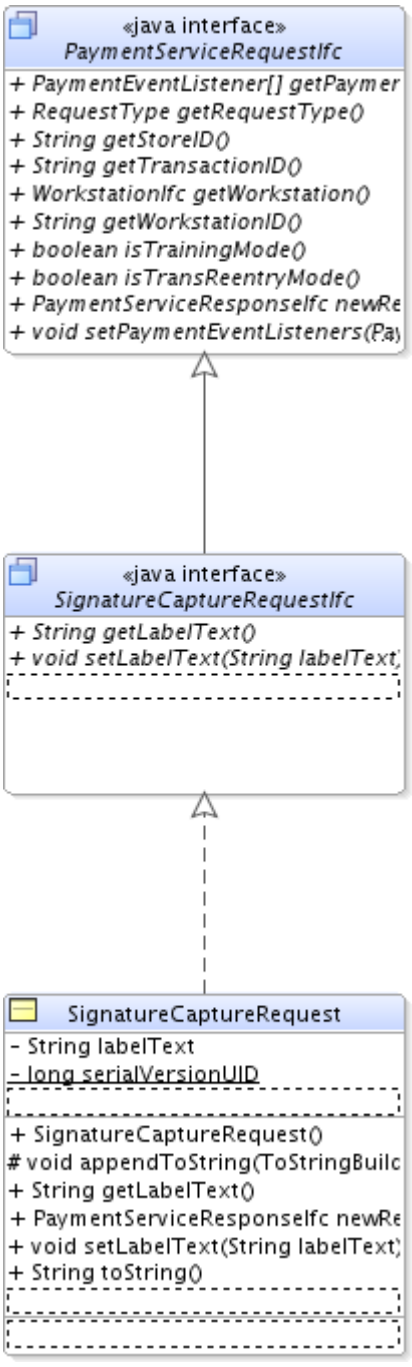
CallRefferalRequest inherits from some of the classes used by the authorize sale request. Those classes are excluded here to simplify the diagram. See Figure 13–1, "AuthorizeTransferRequest Class" for more detail. This request is used when a call referral is required for authorization.

Figure 13–4 AuthorizeCallReferralRequest Class

APF Signature Capture Request

This request acquires a signature from the CPOI device.

Figure 13–5 SignatureCaptureRequest Class



APF Customer Interaction Request

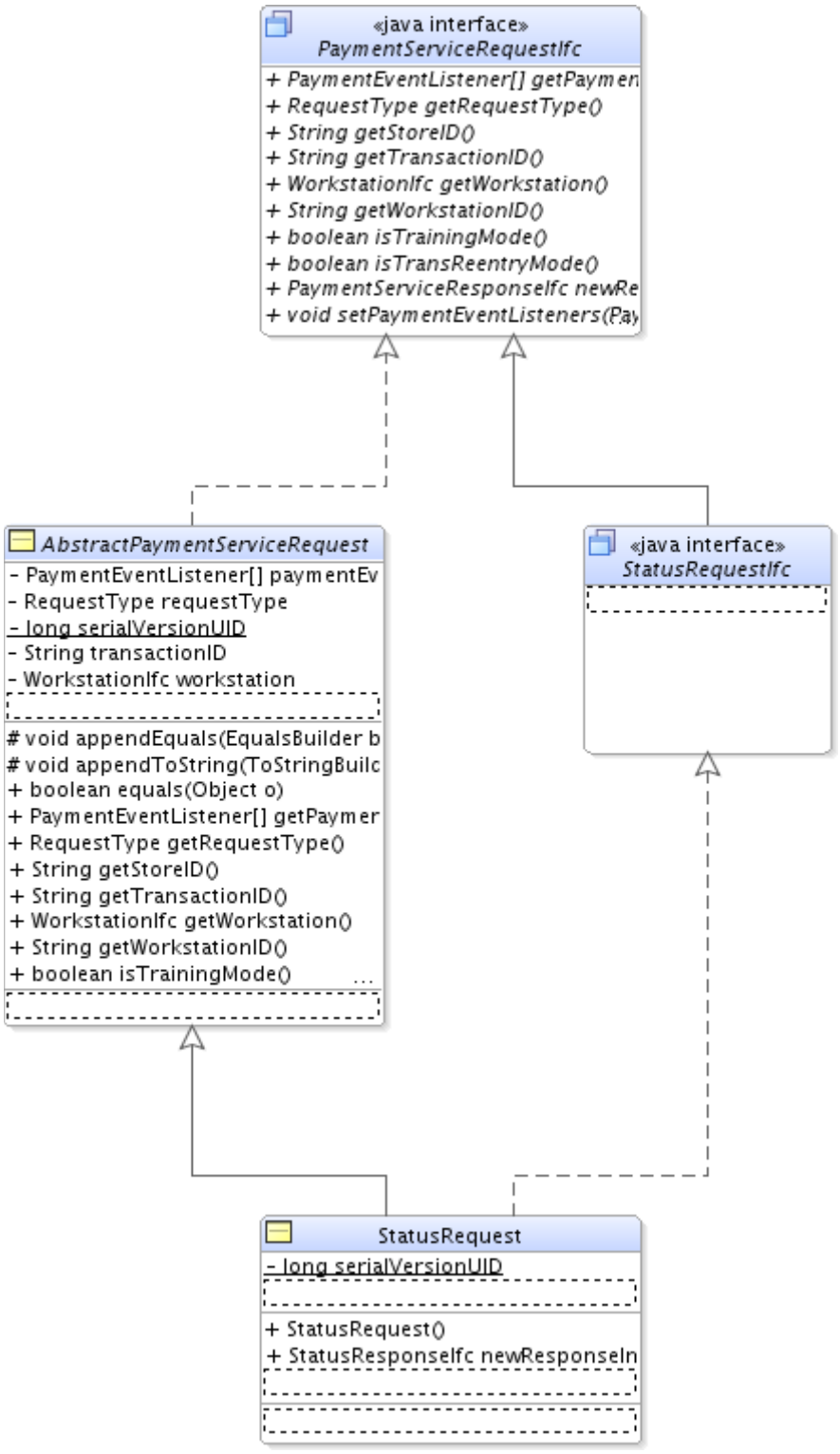
This request is used to display information such as purchased items and tenders on the CPOI device. This request also controls activation and detection of swipe-ahead capability.

Figure 13–6 CustomerInteractionRequest Class

APF Status Request

This request gets the status (online or offline) of the authorization service.

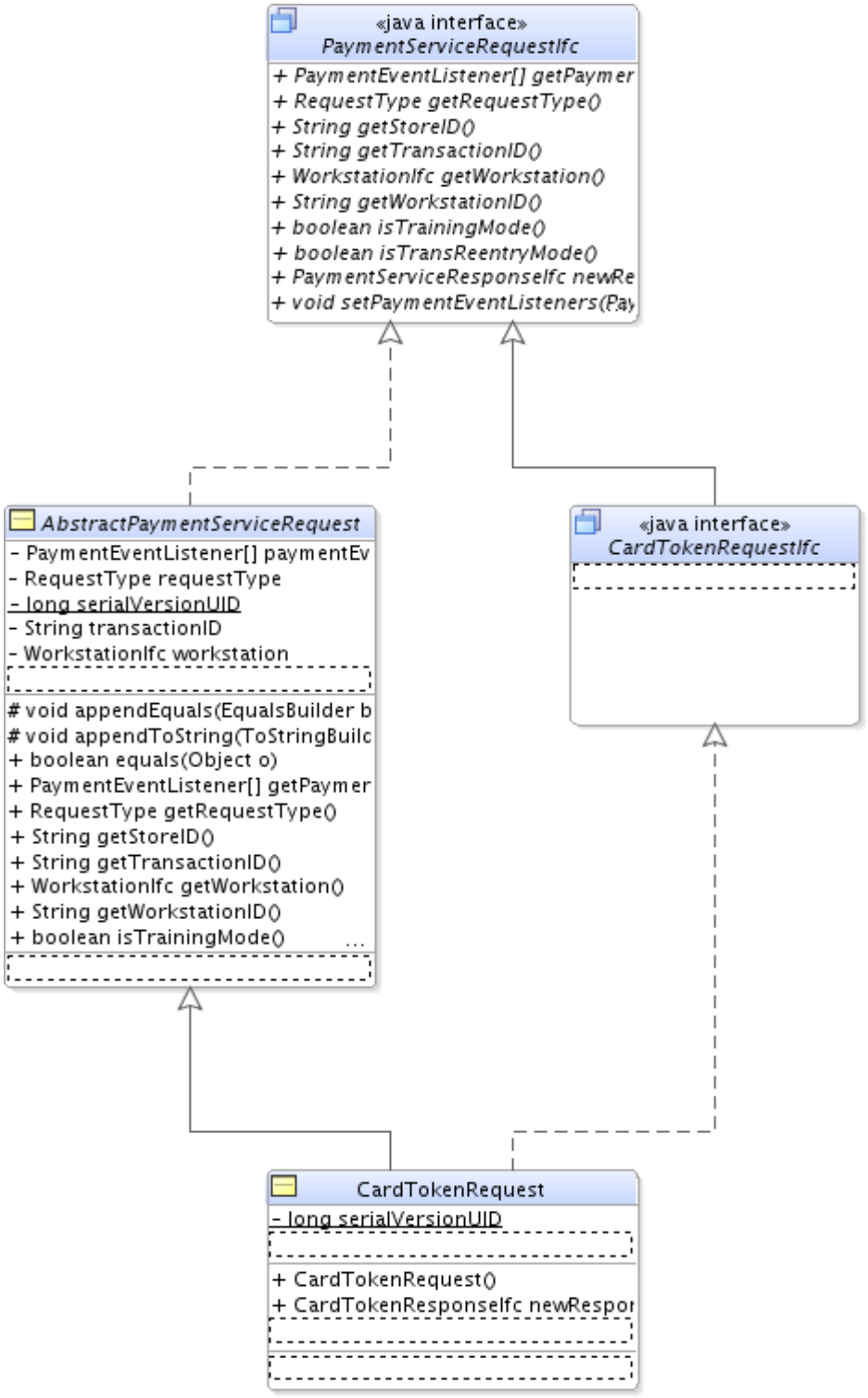
Figure 13–7 StatusRequest Class



APF Get Card Token Request

This request gets a card token from the authorization service. Under some circumstances a card token or masked card number is required without the need to perform any authorization with the bank.

Figure 13–8 CardTokenRequest Class



APF Response Types

The APF response types follow a parallel hierarchy with their request counterparts.

Calling PaymentManger from Point-of-Service Tours (Services)

PaymentManagerIfc is the API used to send all requests to the authorization service (PinComm, PC-EFT, or training mode auth simulator).

PaymentManger uses COMMEXT to route requests to the appropriate technician.

PaymentManager accepts the request objects defined earlier.

The following tours (services) call PaymentManager:

- instantcredit – paymentMgr.authorize(AuthorizeInstantCreditRequestIfc)
- main – paymentManager.clearSwipeAheadData()
- manager – paymentManager.getStatus()
- returns.returnoptions – paymentManager.getCardToken(CardTokenRequestIfc)
- sale – paymentManager.clearSwipeAheadData()
- signaturecapture – paymentManager.getSignature(SignatureCaptureRequestIfc)
- tender – paymentManager.isSwipeAhead()
- tender.activation – paymentManager.authorize(AuthorizeTransferRequestIfc)
- tender.authorization
  - paymentManager.authorize(AuthorizeTransferRequestIfc)
  - paymentManager.authorize(AuthorizeCallReferralRequestIfc)
- tender.reversal - paymentManager.reversal(ReversalRequestIfc)

CPOIPaymentUtility

The CPOIPaymentUtility is a wrapper used primarily to send scrolling receipt requests to the PaymentManager. This utility translates information from Point-of-Service objects into CustomerInteractionRequestIfc objects that are passed to paymentManager.show(). The CPOIPaymentUtility is called from any tour (service) that clears, adds or updates line item or tender information on the CPOI device.

If the new authorization service does not support scrolling receipts, the COMMEXT configuration can be modified to filter these types of requests.

PinComm Technician

This section describes the APF implementation for the PinComm authorization service.

As required by the APF, the PinComm technician uses COMMEXT to route messages to a variety of connectors and formatters which send requests to and receive requests from PinComm.

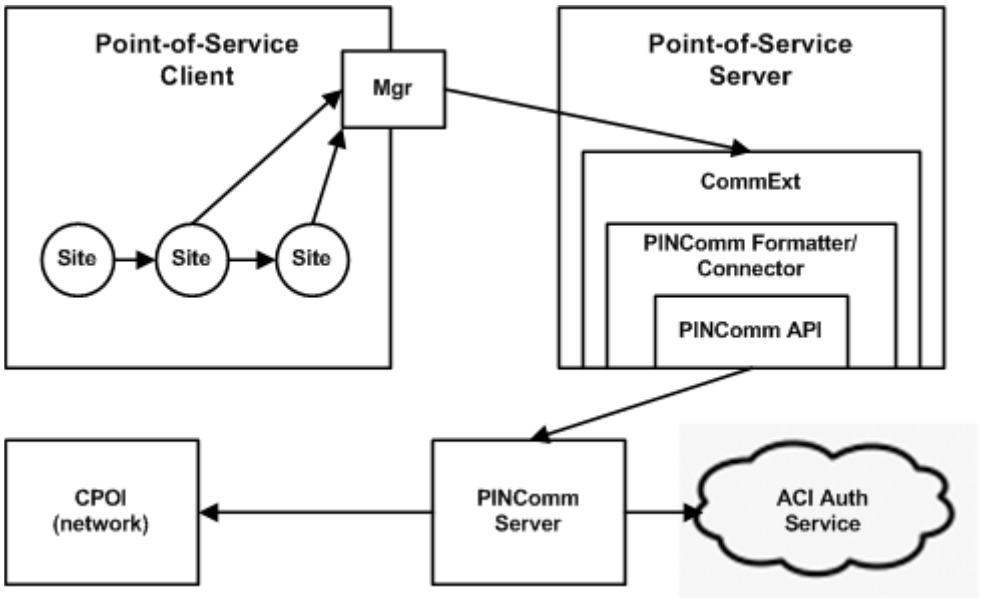
Example Topology

This configuration is an example of an authorization service that has a single central service for a store.

The base COMMEXT configuration for PinComm routes requests from the Point-of-Service client to the Point-of-Service server, which then sends requests to a PinComm server.

Note: Check with your authorization service vendor for the recommended topology. You might want to configure APF so that requests are not sent to the Point-of-Service server. By decoupling the authorization function from the Point-of-Service server, transaction authorizations can continue if the Point-of-Service server is not available to the client.

Figure 13–9 PinComm Topology



PinComm Connectors

The PinComm implementation has several formatters that perform the operations required by the various types of requests.

PinCommConnector

This connector formats, sends and translates. This is the simplest connector and is used when no special processing or routing is required by a request.

PinComm CardAuthConnector

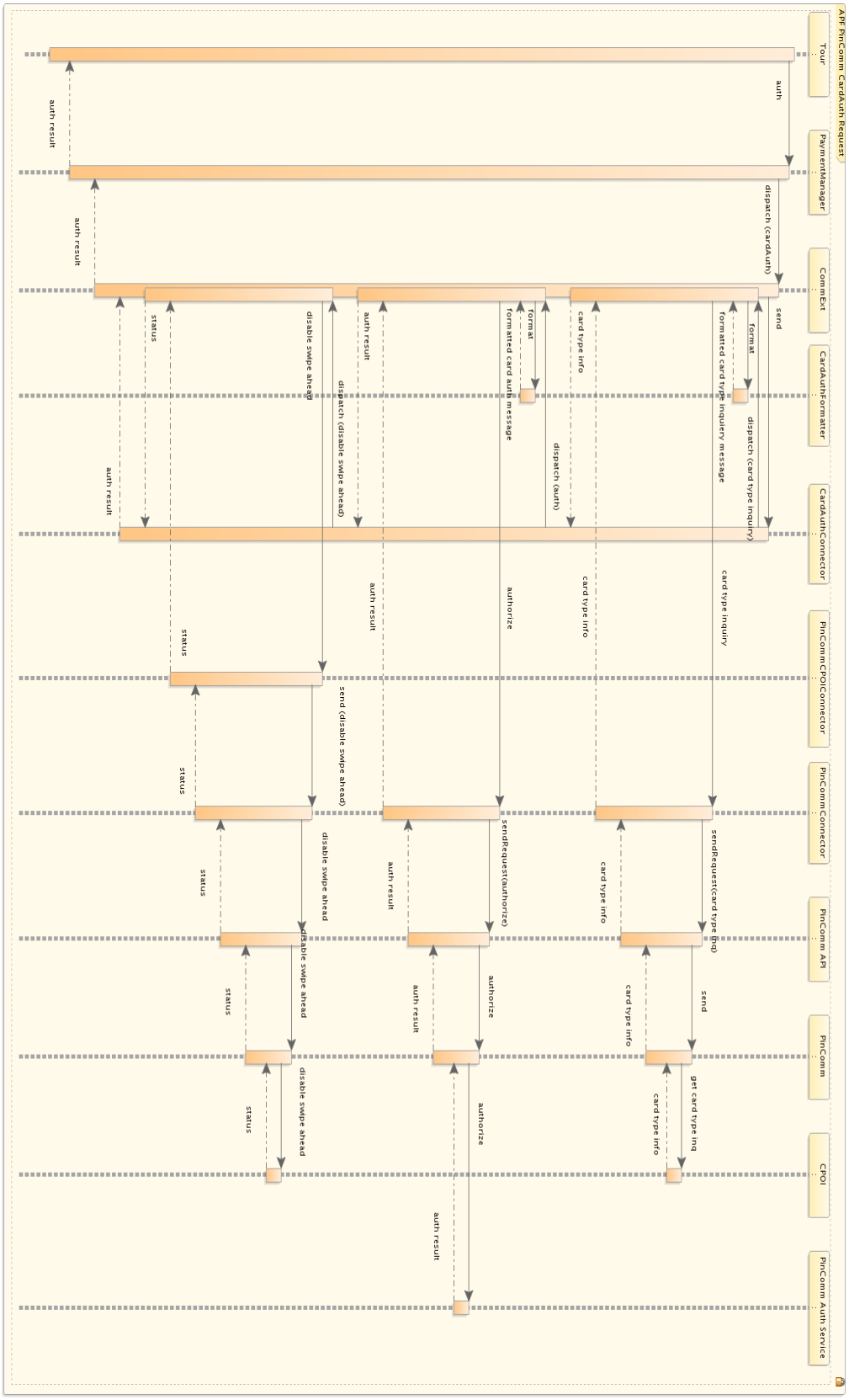
This connector is used primarily for credit, debit and gift card authorizations. A special connector is required because this request is performed in two stages. Stage 1 prompts the customer for the card type, cash back amount (when appropriate), and the amount approval. Stage 2 performs the authorization request. This connector also performs extra processing for the swipe-ahead feature.

The following sequence diagram describes the flow for the CardAuth request. This is the most complex request sequence.

This diagram illustrates the following:

- The ability to dispatch COMMEXT messages from within a connector.
- The ability to extend COMMEXT with a custom connector to handle flows that the COMMEXT configuration settings cannot handle.

Figure 13–10 CardAuthConnector Request



PinComm OnePassCardAuthConnector

This connector performs the format, send and translate operations for the OnePassCardAuth request . This connector also performs extra processing for the swipe-ahead feature. This connector is used when an authorization is performed without the need to prompt the customer for additional information.

PinComm AuthorizeCallReferralWithoutTokenConnector

This connector performs the format, send and translate operations for the AuthorizeCallReferralWithoutToken request. It also performs extra processing for the swipe-ahead feature.

PinComm StatusInquiryConnector

This connector does not communicate with the PinComm server. It checks the internal online and offline flags and returns the appropriate response.

PinComm PinCommCPOIConnector

This connector includes extra processing required for scrolling receipts.

PinComm CardTokenInquiryConnector

This connector performs the format, send and translate operations for the GetCardToken request. This connector also performs extra processing for the swipe-ahead feature.

PinComm ReentryAuthConnector

This connector does not communicate with the PinComm server. This connector creates a response based on the provided request for gift cards and checks.

PinComm Formatters

PinComm has several formatters that perform the operations required by the various types of requests. These formatters translate the APF request objects into a format used by PinComm and translates the PinComm responses into the APF response objects.

Each type of request has its own formatter.

All formatters inherit from the AbstractPinCommFormatter class. The following formatters have more than one level of inheritence.

PinComm CardAuthFormatters

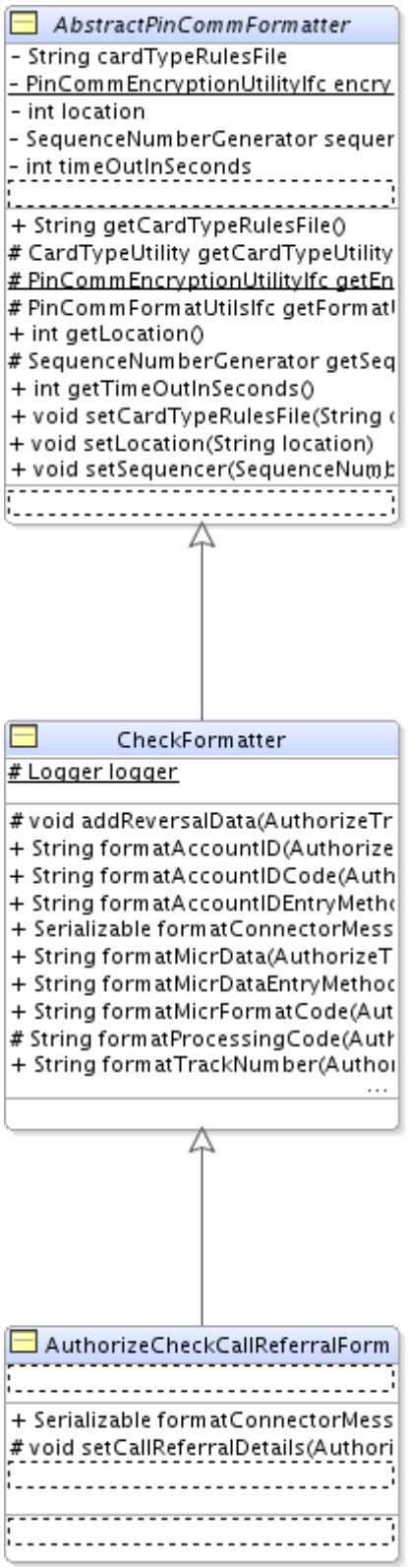
The following formatters translate the various types of card authorization requests.

Figure 13–11 CardAuthFormatters

PinComm Check Formatters

The following classes translate check-related requests.

Figure 13–12 Check Formatters



PinComm Configuration

Many values are configurable in both the APF PinComm technician and the ACI PinComm software. Configurable values for the APF PinComm technician are defined in `<source_directory>/applications/pos/deploy/shared/config/PinCommCodes.properties`. Many of the values in PinCommCodes.properties must match with values configured in the ACI PinComm installation. The ACI PinComm vendor documentation for configuration details.

Servebase PC-EFT

The integration between Point-of-Service and Servebase PC-EFT is accomplished using the Communication Extension (COMMEXT) framework. For Servebase integration, there is only a Manager, with no Technician. The Manager is a COMMEXT variety that knows how to route messages to the Servebase formatter/connector pair. The formatters and connectors split the responsibilities for formatting messages to external systems. The formatters and connectors also split the responsibilities for connecting to those systems.

JAXBFormatter

Abstract base class that is used for formatters that generate and parse XML using JAXB. The class caches JAXB contexts for subsequent use.

Figure 13–13 JAXBFormatter

ServebaseFormatter

Extends the JAXBFormatter class and is responsible for converting to or from Servebase XML. Superclass routines are used to convert the XML into Java objects generated from JAXB. This is the most likely class to extend to provide custom behavior. The methods `formatConnectorMessage(MessageIfc)` and `translateConnectorResponse(Serializable)` are the two points of interaction with an instance of this class and the dispatcher.

ChainedConnector

Abstract connector class that performs some bit of logic, then delegates its message to another connector. In this integration, the chained-to connector is a SocketConnector.

ServebaseConnector

Extends the abstract ChainedConnector and delegates its message to the SocketConnector. ServebaseConnector provides some simple logic to ignore requests from Point-of-Service for device status updates and signature captures, neither of which is supported by Servebase PC-EFT. Extending this class was not intended.

SocketConnector

Given a host name and port number, this connector does the leg work of opening a socket and reading and writing strings to and from PC-EFT. Default configuration for this connector is to connect to a localhost at port 5000. Extending this class should not be required.

SocketThread

Inner class of SocketConnector that monitors the socket connection for a response.

Figure 13–14 APF Flow Diagram

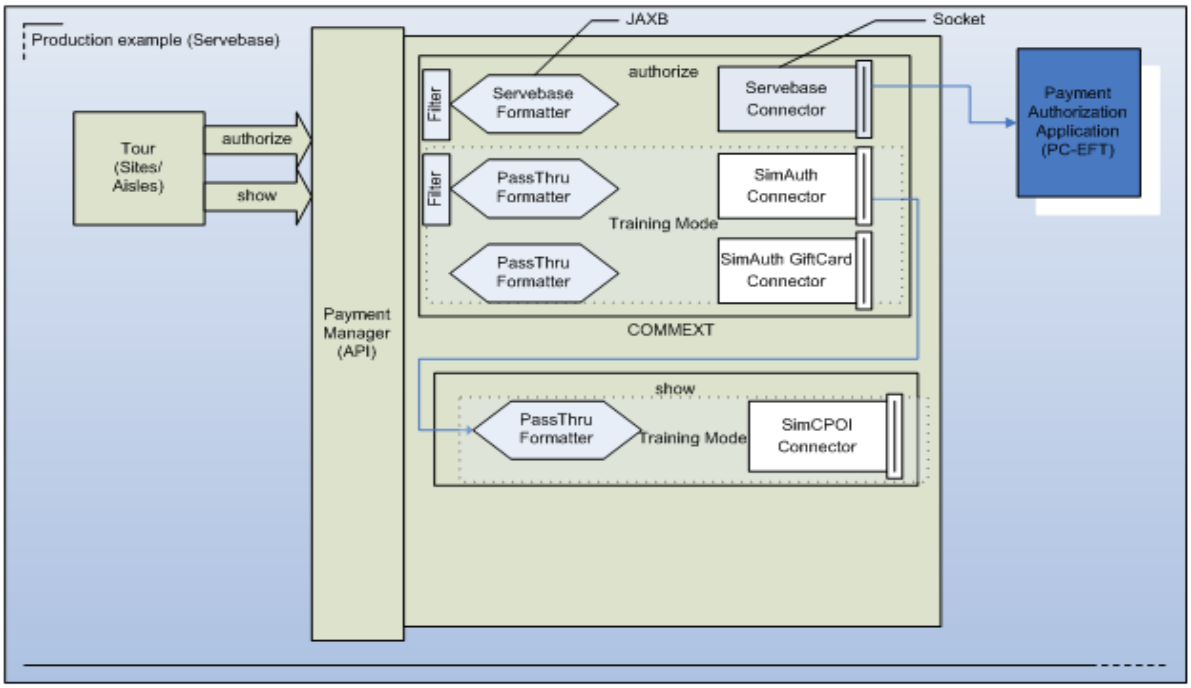


Figure 13–14 shows how tour sites and aisles of the Point-of-Service application communicates to PC-EFT. The tour code communicates through a Manager interface as normal. That Manager uses a COMMEXT Dispatcher to route the messages through routes configured for it. The routes are defined for the actions that the tour is trying to perform, for example, authorize an amount for tender or show some information about the sale on the CPOI. The COMMEXT routing also uses filters in this case to redirect these messages while in training mode to separate connectors along the same route.

Configuration

The COMMEXT framework is used to provide a pluggable and fully configurable integration between Point-of-Service and Servebase PC-EFT. Only the Point-of-Service client integrates with PC-EFT. The PaymentManager is defined in the `<source_directory>/applications/pos/deploy/client/config/conduit` file:

Example 13–1 PaymentManager in pos/config/conduit/ClientConduit.xml

```
<MANAGER name="PaymentManager"
         class="PaymentManager"
         package="oracle.retail.stores.domain.manager.payment"
         export="N" saveValets="N" singleton="N">
    <PROPERTY propname="configScript"
              propvalue="classpath://config/manager/PaymentManager.xml"/>
</MANAGER>
```

The PaymentManager is a COMMEXT BaseManager with its own configuration file, PaymentManager.xml, found in the `<source_directory>/applications/pos/deploy/client/config/manager` directory. In the PaymentManager.xml file, the message routing components are defined and configured to communicate to PC-EFT. To generate a custom behavior, extend the ServebaseFormatter and replace it in this configuration.

- FILTERS – In the group of filters, there are special filters for handling messages differently in training mode and re-entry mode, and filters for handling gift card requests.
- FORMATTERS
  - PassThruFormatter – The most common formatter, PassThruFormatter is defined and used for training mode. PassThruFormatter does not change the contents of the message before it arrives at a Connector.
  - ConnectorValetFormatter – This formatter is not used for Servebase integration.
  - ServebaseFormatter – This is a subclass of a JAXBFormatter. ServebaseFormatter performs the primary work of converting PC-EFT XML messages into response objects and vice versa. ServebaseFormatter must also be configured with the correct merchantId, customerCode, site, username and password to communicate with PC-EFT correctly.
- RULES
  - RetryRule – The number of retries can be configured.
  - StopOnErrorRule – Stops sending the message to connectors upon an error.
  - DefaultActionRule – The routing rule with a default action (Continue) that occurs upon error.

- CONNECTORS
  - SimulatedAuthConnector and the other simulated connectors are used to generated responses while in training mode.
  - ServebaseSocketConnector – This connector does the actual work of opening a socket to PC-EFT and sending or receiving the XML strings. ServebaseSocketConnector should have the hostName and port configuration set to point to where Servebase PC-EFT is installed. The base product expects PC-EFT to be in the localhost. The connector should be configured to expect a response from PC-EFT:

    ```
    expectResponse = true
    ```

  - ServebaseConnector – This connector is where most messages are routed. ServebaseConnector provides some logic for ignoring some device status requests and rejecting unsupported requests, such as GiftCard and HouseAccount. This connector forwards valid requests to the ServebaseSocketConnector.
  - ReversalConnector – A file-based queue connector for forwarding the reversal requests to PC-EFT in a non-synchronous fashion. The interval configurations are in milliseconds.
- MSGROUTERS
  - DEFAULTROUTER – This is the default route that all messages sent from Point-of-Service flow through when a match MSGROUTER type is not found for the message being dispatched. Requests made in training mode are filtered off and handled by the simulated connectors.
  - Reversals are ignored in training mode and go to the store-and-forward connector, ReversalConnector, in normal mode.

Message Formats

PC-EFT expects communication in XML format. Refer to Servebase's PC-EFT POS (FIXED PED) XML Integration Guide for details about the format of the communication between Point-of-Service and PC-EFT.

Example 13–2 Request Format

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<IccTransactionRequest xmlns="http://servebase.com/2009/06/pedframework">
    <TransactionConfig>
        <CustomerCode>ORA</CustomerCode>
        <Site>ORA000000001</Site>
        <Culture>en</Culture>
        <Workstation>001</Workstation>
        <MerchantId>21249872</MerchantId>
        <Username>ORA-001</Username>
        <Password>_F4Rvcf-G</Password>
        <IpAddress>127.0.0.1</IpAddress>
    </TransactionConfig>
    <AuthorizationConnectionType>OnlineAuthorization</AuthorizationConnectionType>
    <TransactionAmount currency="GBP">10.00</TransactionAmount>
    <TransactionReference>042411290016</TransactionReference>
    <TransactionDateTime>2010-11-10T10:10:10.000-06:00</TransactionDateTime>
    <TransactionType>Sale</TransactionType>
</IccTransactionRequest>
```

Although the above example is formatted for readability, the actual XML produced is not formatted.

Example 13–3 Response Format

```
<?xml version="1.0" encoding="utf-16"?>
<IccTransactionResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://servebase.com/2009/06/pedframework">
  <ResponseCode>Approved</ResponseCode>
  <ResponseMessage>Transaction Approved</ResponseMessage>
  <ReceiptInfo xsi:type="IccReceiptInfo">
    <CardNumber>541333******0020</CardNumber>
    <ExpiryDate year="14" month="12" />
    <StartDate year="4" month="1" />
    <TransactionAmount currency="GBP">20.00</TransactionAmount>
    <TransactionReference>042411290016</TransactionReference>
    <MerchantId>21249872</MerchantId>
    <TerminalId>27519414</TerminalId>
    <CardScheme code="MSC"
creditDebitIndicator="CreditCard">Mastercard</CardScheme>
    <CaptureMethod>Icc</CaptureMethod>
    <Dcc xsi:nil="true" />
    <ApplicationId>A0000000041010</ApplicationId>
    <ApplicationLabel>MasterCard</ApplicationLabel>
    <PreferredName>MasterCard</PreferredName>
    <PanSequenceNumber>00</PanSequenceNumber>
    <CvmResults>410302</CvmResults>
    <TransactionType>Sale</TransactionType>
    <IccAccreditationInfo>

<AuthorisationRequestCryptogram>CB4732891FAF9FEF</AuthorisationRequestCryptogram>
      <ApplicationInterchangeProfile>5800</ApplicationInterchangeProfile>
      <ApplicationTransactionCounter>0008</ApplicationTransactionCounter>
      <UnpredictableNumber>7220921C</UnpredictableNumber>
      <TerminalVerificationResult>0000008000</TerminalVerificationResult>
      <CryptogramTransactionType>00</CryptogramTransactionType>
      <CryptogramInformationData>40</CryptogramInformationData>

<ApplicationResponseCryptogram>DC7AF2C53204A954</ApplicationResponseCryptogram>
      <POSEntryMode1>3</POSEntryMode1>
      <POSEntryMode2>2</POSEntryMode2>
      <ApplicationUsageControl>FF00</ApplicationUsageControl>
      <ApplicationVersionNumber>0002</ApplicationVersionNumber>
      <TerminalApplicationVersionNumber>0002</TerminalApplicationVersionNumber>
      <TransactionStatusInformation>E800</TransactionStatusInformation>
      <TerminalType>22</TerminalType>
      <TerminalCapabilities>E0B8C8</TerminalCapabilities>
      <IssuerActionCodesOnline>F870A49800</IssuerActionCodesOnline>
      <IssuerActionCodesDenial>0000000000</IssuerActionCodesDenial>
      <IssuerActionCodesDefault>FC50A00000</IssuerActionCodesDefault>
```

```
<IssuerApplicationData>021265100F040000DAC000000000000000FF</IssuerApplicationData
>
      <AuthorisationResponseCode>00</AuthorisationResponseCode>
      <TerminalCountryCode>0826</TerminalCountryCode>
      <TerminalCurrencyNumber>826</TerminalCurrencyNumber>
    </IccAccreditationInfo>
  </ReceiptInfo>
  <Token>7a0351bf-277f-4340-a811-0ab026e886b8</Token>
  <AuthorityCode>006375</AuthorityCode>
  <IssuerAuthenticationData>DC7AF2C53204A9540012</IssuerAuthenticationData>
</IccTransactionResponse>
```

Response Codes

The following are some notes about the translation of response code for Servebase.

- If PC-EFT responds with an ERROR or DECLINED response code, and the message `Failed to communicate`, then Point-of-Service treats that as a Referral response and follows the appropriate flow.
- If PC-EFT responds with a CANCELLED response code, and the message `did not response in the configured time span`, then Point-of-Service treats that as a Timeout and follows the appropriate flow.
- The PC-EFT CaptureMethod KEYED is translated as the Point-of-Service EntryMethod Swiped.

Training Mode

In training mode, authorization request messages from Point-of-Service are intercepted by filters in the COMMEXT configuration. The requests are redirected to code that provides simulated responses. Depending on the digit in the ones column authorization amount request, the system provides a different response (for reference, $12.34 has a 2 in the ones column).

14

Intra Store Data Distribution Infrastructure

The Oracle Retail Point-of-Service client needs the following producers and consumers datasets to support offline functionality:

- Employee
- Item
- Advanced Pricing
- Tax
- Currency
- Store Info
- Merchandise Hierarchy
- Shipping Method
- Reason Codes
- Discount
- ExportDB

Intra Store Data Distribution Infrastructure (IDDI) automates the following:

- DataSet file generation at the Point-of-Service server
- DataSet file transfer from Point-of-Service server to Point-of-Service client
- Importing dataset files to Point-of-Service client database

Spring Configuration

The system has been designed to support a pluggable model. The following are all designed to be configurable at deployment time:

- DataSetProducerJob
- ClientDataSetController
- DataSetService
- ClientDataSetService
- DataSetProducers
  - StoreInfoDataSetProducer
  - AdvancedPricingDataSetProducer
  - CurrencyDataSetProducer

  - EmployeeDataSetProducer
  - ItemDataSetProducer
  - MerchandiseDataSetProducer
  - OfflineDBProducer
  - ReasonCodeDataSetProducer
  - ShippingMethodDataSetProducer
  - TaxDataSetProducer
  - DiscountDataSetProducer
- DataSetConsumers
  - StroreInfoDataSetConsumer
  - AdvancedPricingDataSetConsumer
  - CurrencyDataSetConsumer
  - EmployeeDataSetConsumer
  - ItemDataSetConsumer
  - MerchandiseDataSetConsumer
  - OfflineDBConsumer
  - ReasonCodeDataSetConsumer
  - ShippingMethodDataSetConsumer
  - TaxDataSetConsumer
  - DiscountDataSetConsumer
- DerbyDataFormatter

This configuration is accomplished through the use of the Spring Framework as a configuration framework.

Table 14–1 includes the set of Spring bean IDs used for each of the pluggable components.

Table 14–1 Spring Framework Configuration Options

Spring bean ID	Purpose	Provided implementation	Configurable Options
service_DataSetService	Configuration for DataSetService. Contains the list of all the DataSetKeys.	oracle.retail.stores. foundation.iddi.DataSetService	Generate at start up. To add a new DataSet type, add one more service_config_<<DataSetType>_KEY
service_ ClientDataSetService	Configuration for ClientDataSetService. Contains the list of all the DataSetKeys.	oracle.retail.stores. foundation.iddi.ClientDataSetService	To add a new DataSet type, add one more service_config_<<DataSetType>_KEY dataImportFilePath(service_config_DataImportFilePath)
service_ FrequentProducerJob	Producer Job that runs frequently. Configured to run once every 15 minutes by default.	org.springframework.scheduling.quartz.JobDetailBeanservice_DataSetService	To add a new DataSet type, add one more service_config_<<DataSetType>_KEY

Table 14–1 (Cont.) Spring Framework Configuration Options

Spring bean ID	Purpose	Provided implementation	Configurable Options
service_ InfrequentProducerJob	Producer Job configured to run once a day by default.	org.springframework. scheduling.quartz.JobDetailBeanservice_DataSetService	To add a new DataSet type, add one more service_config_ <<DataSetType>_KEY
service_ OfflineDBProducerJob	Producer Job configured to run once a day by default.	org.springframework. scheduling.quartz.JobDetailBeanservice_DataSetService	To add a new DataSet type, add one more service_config_ <<DataSetType>_KEY
service_ TriggerFrequentProducer	Cron Job Trigger class that runs service_ FrequentProducerJob configuration. Cron Expression value can be modified to configure the job frequency.Cron Expression format.value="0 0,15,30,45 * * * ?" Value parameters from left to right separated by spaceSecondsMinutes HoursDaysWeeksYear s To configure more than one value to any of the value parameter, configure values separated by commas (,) * Indicates any value	org.springframework. scheduling.quartz.CronTriggerBean	service_FrequentProducerJob Cron Expression Value
service_ TriggerInfrequentProducer	Trigger class that runs service_ InfrequentProducerJob configuration	org.springframework. scheduling.quartz.CronTriggerBean	service_ InfrequentProducerJob Cron Expression Value service_ ProducerSchedulerFactory Registers the services, service_ TriggerFrequentProducerservice_ TriggerInfrequentProducer with the Quartz SchedulerFactoryBean org.springframework. scheduling.quartz.SchedulerFactoryBeanservice_ TriggerFrequentProducerservice_ TriggerInfrequentProducer
service_ TriggerOfflineDBProducer	Trigger class that runs service_ OfflineProducerJob configuration	org.springframework. scheduling.quartz.CronTriggerBean	service_ OfflineDBProducerJob Cron Expression Value

Table 14–1 (Cont.) Spring Framework Configuration Options

Spring bean ID	Purpose	Provided implementation	Configurable Options
service_CurrencyProducer	DataSet Key definition for Currency DataSetProducer	oracle.retail.stores.domain.iddi.CurrencyDataSetProducer	dataSetKey (service_config_CUR_KEY)dataExportFilePath (service_config_DataExportFilePath)dataExportZipFilePath (service_config_DataExportZipFilePath)fileWriter(service_FileWriter)
service_TaxProducer	DataSet Key definition for Tax DataSetProducer	oracle.retail.stores.domain.iddi.TaxDataSetProducer	dataSetKey(service_config_TAX_KEY)dataExportFilePath (service_config_DataExportFilePath)dataExportZipFilePath (service_config_DataExportZipFilePath)fileWriter(service_FileWriter)
service_EmployeeProducer	DataSet Key definition for Employee Producer	oracle.retail.stores.domain.iddi.EmployeeDataSetProducer	dataSetKey(service_config_EMP_KEY)dataExportFilePath (service_config_DataExportFilePath)dataExportZipFilePath (service_config_DataExportZipFilePath)fileWriter(service_FileWriter)
service_AdvancedPricingProducer	DataSet Key definition for Advanced Pricing DataSetProducer	oracle.retail.stores.domain.iddi.PricingDataSetProducer	dataSetKey(service_config_PRC_KEY)dataExportFilePath (service_config_DataExportFilePath)dataExportZipFilePath (service_config_DataExportZipFilePath)fileWriter(service_FileWriter)
service_ItemProducer	DataSet Key definition for Item DataSetProducer	oracle.retail.stores.domain.iddi.ItemDataSetProducer	dataSetKey(service_config_ITM_KEY)dataExportFilePath (service_config_DataExportFilePath)dataExportZipFilePath (service_config_DataExportZipFilePath)fileWriter(service_FileWriter)
service_StoreInfoProducer	DataSet Key definition for Store Info Producer	oracle.retail.stores.domain.iddi.StroreInfoDataSetProducer	dataSetKey(service_config_STORE_KEY)dataExportFilePath (service_config_DataExportFilePath)dataExportZipFilePath (service_config_DataExportZipFilePath)fileWriter(service_FileWriter)
service_MerchandiseProducer	DataSet Key definition for Merchandise Hierarchy Producer	oracle.retail.stores.domain.iddi.MerchandiseHierarchyDataSetProducer	dataSetKey(service_config_MER_KEY)dataExportFilePath (service_config_DataExportFilePath)dataExportZipFilePath (service_config_DataExportZipFilePath)fileWriter(service_FileWriter)

Table 14–1 (Cont.) Spring Framework Configuration Options

Spring bean ID	Purpose	Provided implementation	Configurable Options
service_ ShippingMethodProducer	DataSet Key definition for Shipping Method Producer	oracle.retail.stores. domain.iddi.ShippingMethod DataSetProducer	dataSetKey(service_config_ SHP_MTH_ KEY)dataExportFilePath (service_config_ DataExportFilePath)dataExpo rtZipFilePath (service_config_ DataExportZipFilePath)fileWr iter(service_FileWriter)
service_ ReasonCodeProducer	DataSet Key definition for Reason Codes Producer	oracle.retail.stores. domain.iddi.ReasonCodesDat aSetProducer	dataSetKey(service_config_ RSN_CODE_ KEY)dataExportFilePath (service_config_ DataExportFilePath)dataExpo rtZipFilePath (service_config_ DataExportZipFilePath)fileWr iter(service_FileWriter)
service_DiscountProducer	DataSet Key definition for Discount Producer	oracle.retail.stores. domain.iddi.DiscountDataSet Producer	dataSetKey(service_config_ DISCOUNT_ KEY)dataExportFilePath (service_config_ DataExportFilePath)dataExpo rtZipFilePath (service_config_ DataExportZipFilePath)fileWr iter(service_FileWriter)
service_ OfflineDBProducer	DataSet Key definition for OfflineDB Producer	oracle.retail.stores. domain.iddi.OfflineDBDataSe tProducer	dataSetKey(service_config_ OFFLINEDB_ KEY)dataExportFilePath (service_config_ DataExportFilePath)dataExpo rtZipFilePath (service_config_ DataExportZipFilePath)fileWr iter(service_FileWriter)
service_ CurrencyConsumer	DataSet Key definition for Currency DataSetConsumer	oracle.retail.stores. domain.iddi.CurrencyDataSet Consumer	dataSetKey(service_config_ CUR_ KEY)dataImportFilePath(serv ice_config_ DataImportFilePath)importH elper(service_ OfflineDBHelper)
service_TaxConsumer	DataSet Key definition for Tax DataSetConsumer	oracle.retail.stores. domain.iddi.TaxDataSetCons umer	dataSetKey(service_config_ TAX_ KEY)dataImportFilePath(serv ice_config_ DataImportFilePath)importH elper(service_ OfflineDBHelper)
service_ EmployeeConsumer	DataSet Key definition for Employee DataSetConsumer	oracle.retail.stores. domain.iddi.EmployeeDataSe tConsumer	dataSetKey(service_config_ EMP_ KEY)dataImportFilePath(serv ice_config_ DataImportFilePath)importH elper(service_ OfflineDBHelper)

Table 14–1 (Cont.) Spring Framework Configuration Options

Spring bean ID	Purpose	Provided implementation	Configurable Options
service_ AdvancedPricingConsumer	DataSet Key definition for Advanced Pricing DataSetConsumer	oracle.retail.stores. domain.iddi.AdvancedPricingDataSetConsumer	dataSetKey(service_config_ PRC_ KEY)dataImportFilePath(service_config_ DataImportFilePath)importHelper(service_ OfflineDBHelper)
service_ItemConsumer	DataSet Key definition for Item DataSetConsumer	oracle.retail.stores. domain.iddi.ItemDataSetConsumer	dataSetKey(service_config_ ITM_ KEY)dataImportFilePath(service_config_ DataImportFilePath)importHelper(service_ OfflineDBHelper)
service_ StoreInfoConsumer	DataSet Key definition for Store Info DataSetConsumer	oracle.retail.stores. domain.iddi.StoreInfoDataSetConsumer	dataSetKey(service_config_ STORE_ KEY)dataImportFilePath(service_config_ DataImportFilePath)importHelper(service_ OfflineDBHelper)
service_ MerchandiseConsumer	DataSet Key definition for Merchandise Hierarchy DataSetConsumer	oracle.retail.stores. domain.iddi.MerchandiseHierarchyDataSetConsumer	dataSetKey(service_config_ MER_ KEY)dataImportFilePath(service_config_ DataImportFilePath)importHelper(service_ OfflineDBHelper)
service_ ShippingMethodConsumer	DataSet Key definition for Shipping Method DataSetConsumer	oracle.retail.stores. domain.iddi.ShippingMethodDataSetConsumer	dataSetKey(service_config_ SHP_MTH_ KEY)dataImportFilePath(service_config_ DataImportFilePath)importHelper(service_ OfflineDBHelper)
service_ ReasonCodesConsumer	DataSet Key definition for Reason Codes DataSetConsumer	oracle.retail.stores. domain.iddi.ReasonCodesDataSetConsumer	dataSetKey(service_config_ RSN_CODE_ KEY)dataImportFilePath(service_config_ DataImportFilePath)importHelper(service_ OfflineDBHelper)
service_ DiscountConsumer	DataSet Key definition for Discount DataSetConsumer	oracle.retail.stores. domain.iddi.DiscountDataSetConsumer	dataSetKey(service_config_ DISCOUNT_ KEY)dataImportFilePath(service_config_ DataImportFilePath)importHelper(service_ OfflineDBHelper)
service_ OfflineDBConsumer	DataSet Key definition for OfflineDB DataSetConsumer	oracle.retail.stores. domain.iddi.OfflineDBDataSetConsumer	dataSetKey(service_config_ OFFLINEDB_ KEY)dataImportFilePath(service_config_ DataImportFilePath)importHelper(service_ OfflineDBHelper)

Table 14–1 (Cont.) Spring Framework Configuration Options

Spring bean ID	Purpose	Provided implementation	Configurable Options
service_ OfflineDBConsumerJob	Consumer Job that runs frequently. Configured to run once a day by default	org.springframework.scheduling.quartz.JobDetailBean	dataSets To add a new DataSet type, add one more service_config_ <<DataSetType>_KEY
service_ FrequentConsumerJob	Consumer Job that runs frequently. Configured to run every 15mins by default	org.springframework.scheduling.quartz.JobDetailBean	dataSets To add a new DataSet type, add one more service_config_ <<DataSetType>_KEY
service_ InfrequentConsumerJob	Consumer Job configured to run once a day by default.	org.springframework.scheduling.quartz.JobDetailBean	dataSets To add a new DataSet type, add one more service_config_ <<DataSetType>_KEY
service_ TriggerOfflineDBConsumer	Cron Job Trigger class that runs service_ OfflineDBConsumer configuration.	org.springframework.scheduling.quartz.CronTriggerBean	service_ OfflineDBConsumerJob CronExpression Value
service_ TriggerFrequentConsumer	Cron Job Trigger class that runs service_ FrequentConsumer configuration.	org.springframework.scheduling.quartz.CronTriggerBean	service_ FrequentConsumerJob CronExpression Value
service_ TriggerInfrequentConsumer	Cron Job Trigger class that runs service_ InfrequentConsumer configuration.	org.springframework.scheduling.quartz.CronTriggerBean	service_ InfrequentConsumerJob CronExpression Value
service_ clientSchedulerFactory	Registers the services, service_ TriggerFrequentConsumerservice_ TriggerInfrequentConsumer with the Quartz SchedulerFactoryBean	org.springframework.scheduling.quartz.SchedulerFactoryBean	service_ TriggerFrequentConsumer service_ TriggerInfrequentConsumer
service_config_ DataExportFilePath	Configuration for Data Export File Path. This is the relative path. Application takes the application running path and appends the path given in this configuration.	java.lang.String	value

Table 14–1 (Cont.) Spring Framework Configuration Options

Spring bean ID	Purpose	Provided implementation	Configurable Options
service_config_ DataExportZipFilePath	Configuration for Data Export Zip File Path. This is the relative path. Application takes the application running path and appends the path given in this configuration.Note: The service_config_ DataExportFilePath should not contain DataSetKey names (eg: EMPLOYEE, ITEM, CURRENCY, ADVANCED_ PRICING,TAX)	java.lang.String	value
service_config_ DataImportFilePath	Configuration for Data Import File Path where the dataset files will be downloaded from Point-of-Service server and cached.	java.lang.String	value
service_config_ OfflineSchemaSQLFilePat h	Folder configuration where the Offline database schema SQL File.	java.lang.String	value
service_config_ OfflineSchemaLogFilePath	Folder configuration for storing the Offline database schema SQL File import log file.	java.lang.String	value
service_OfflineDBHelper	Point-of-Service client offline Database Helper Class configuration	oracle.retail.stores.foundation .iddi.OfflineDerbyHelper	dataImportFilePath service_config_ OfflineSchemaSQLFilePath service_config_ OfflineSchemaLogFilePath
service_ ApplicationVersion	Application Version retreival class configuration.	oracle.retail.stores.pos.PosVer sion	None
service_DataFormatter	Data Formatter Helper to format Point-of-Service server data to Derby data import format specifications.	oracle.retail.stores.foundation .iddi.DerbyDataFormatter	None
service_ DerbyDBFormatter	Data Formatter Helper to format Point-of-Service server data to Derby data import format specifications.	oracle.retail.stores.iddi.Derby DBFormatter	None
service_Filewriter	Format Derby import files.	oracle.retail.stores.foundation .iddi.IDDIFileWriter	Formatter

Table 14–1 (Cont.) Spring Framework Configuration Options

Spring bean ID	Purpose	Provided implementation	Configurable Options
service_DerbyWriter	Bean used to insert data read from store server database into server side offline derby database.	oracle.retail.stores.iddi.IDDIDerbyWriter	■ Formatter. service_DerbyDBFormatter ■ Location of the schema file for import. service_config_OfflineSchemaSQLFilePath ■ Location of the log file to use during import. service_config_OfflineSchemaLogFilePath
service_config_EMP_KEY	DataSet key Configuration	java.lang.String	None
service_config_CUR_KEY	DataSet key Configuration	java.lang.String	None
service_config_TAX_KEY	DataSet key Configuration	java.lang.String	None
service_config_ITM_KEY	DataSet key Configuration	java.lang.String	None
service_config_PRC_KEY	DataSet key Configuration	java.lang.String	None
service_config_MER_KEY	DataSet key Configuration	java.lang.String	None
service_config_SHP_MTH_KEY	DataSet key Configuration	java.lang.String	None
service_config_RSN_CODE_KEY	DataSet key Configuration	java.lang.String	None
service_config_STORE_KEY	DataSet key Configuration	java.lang.String	None
service_config_DISCOUNT_KEY	DataSet key Configuration	java.lang.String	None
service_config_OFFLINEDB_KEY	DataSet key Configuration	java.lang.String	None

For Point-of-Service, the ServiceContext.xml is under `<install directory>\<client or server>\pos\config\context`.

Application Configuration

The timeout interval to start data consumption is configured in the application.xml file. The IDDITimeoutInterval parameter value is set to 15 minutes by default and is configurable.

The IDDIOfflineSupport parameter has been renamed to IDDIOfflineSupportRequired, and the values are reversed. Basically, this parameter allows the end-user to decide if the client should come up without offline data. If IDDIOfflineSupportRequired is **Y**, then the client does not start if no offline data is available (offline data is required for the client to start). If IDDIOfflineSupportRequired is **N**, then the client starts without offline data (offline data is not required for the client to start).

The batch size of the records to write data to offline file is set in domain.properties with the property IDDIBatchSize.

Integration Considerations

IDDI integrates with both the Point-of-Service server and the Point-of-Service client application. IDDI integration with Point-of-Service server produces dataset files on a scheduled basis. IDDI integration with Point-of-Service client downloads the dataset files from Point-of-Service server on a scheduled basis, and the client can then consume those files. IDDI server and client integration is pluggable and configurable.

Point-of-Service client should be online when it is run the first time to download the data from Point-of-Service server. If there is no offline data available, Point-of-Service client does not function in offline mode.

The client-side database schema must be in sync with server-side database schema.

Table 14–2 has been used in Derby database at the Point-of-Service client. The database schema for the following tables must mach the Point-of-Service server database schema.

Table 14–2 Point-of-Service DataSet Table

DataSet Name	DataSet Tables	
Items	AS_ITM	CO_EV
	AS_ITM_I8	TR_CHN_TMP_PRC
	ID_IDN_PS	CO_CLR
	PA_MF	CO_CLR_I8
	PA_MF_I8	CO_SZ
	AS_POG	CO_SZ_I8
	AS_ITM_ASCTN_POG	CO_STYL
	AS_ITM_STK	CO_STYL_I8
	CO_UOM	CO_EV_I8
	CO_UOM_I8	CO_EV_MNT
	ID_DPT_PS	CO_EV_MNT_I8
	ID_DPT_PS_I8	MA_PRC_ITM
	AS_ITM_RTL_STR	MA_ITM_PRN_PRC_ITM
	CO_ASC_RLTD_ITM	MA_ITM_TMP_PRC_CHN
	CO_CLN_ITM	TR_CHN_PRN_PRC
	AS_ITM_SRZ_LB	AS_ITM_SRZ_LB_I8
Employees	PA_EM	CO_ACS_GP_RS_LS
	CO_GP_WRK	CO_ACS_GP_RS_LS_I8
	CO_GP_WRK_I8	PA_RS
	CO_ACS_GP_RS	PA_RS_I8
Advanced Pricing	RU_PRDV	TR_ITM_MXMH_PRDV
	RU_PRDV_I8	CO_EL_PRDV_ITM
	CO_PRDV_ITM	CO_EL_PRDV_DPT
	RU_PRDVC_MXMH	CO_EL_CTAF_PRDV
	CO_PRCGP_I8	CO_EL_MRST_PRDV
	PA_GP_CT	CO_EL_TM_PRDV
	PA_GP_CT_I8	CO_PRCGP
Tax	RU_TX_GP	GEO_TX_JUR
	RU_TX_RT	CO_GP_TX_ITM
	PA_ATHY_TX	CO_GP_TX_ITM_I8
	CO_TX_JUR_ATHY_LNK	PA_TY_TX
	CD_GEO	
Currency	CO_CNY	CO_CNY_DNM
	CO_RT_EXC	CO_CNY_DNM_I8
Store Info	PA_STR_RTL	LO_ADS
	PA_STR_RTL_I8	

Table 14–2 (Cont.) Point-of-Service DataSet Table

DataSet Name	DataSet Tables	
Merchandise Hierarchy	ST_ASCTN_MRHRC	CO_MRHRC_LV
	CO_MRHRC_FNC	CO_MRHRC_LV_I8
	CO_MRHRC_GP	AS_MRHRC_ITM_GP
	CO_MRHRC_GP_I8	
Shipping Method	CO_SHP_MTH	CO_SHP_MTH_I8
Reason Codes	ID_LU_CD	LO_DPT_POS_RTL_STR
	ID_LU_CD_I8	

DataSet Compressed File Structure

The dataset compressed file contains all the dataset flat files of the tables associated with the dataset and metadata information (for example, the Manifest file).

Here is the structure of the dataset compressed file:

```
<DataSet Flat file>
<DataSet Flat file>
<DataSet Flat file>
META-INF\MANIFEST.MF
```

DataSet Compressed File Example

The server generates the compressed file to *<install directory>*\Server\pos\bin\IDDI, and the client copies the compressed file to *<install directory>*\Client\pos\bin\IDDI_CACHE.

The Currency DataSet compressed file (CURRENCY_<<BATCHID>>.ZIP) contains:

```
META-INF\MANIFEST.MF
CO_ACS_GP_RS.TXT
CO_GP_WRK.TXT
PA_RS.TXT
```

Manifest File Structure

The Manifest file compressed in the DataSet compressed files contains dataset metadata information in the following format:

```
DataSetName: <<DataSetName>>
DataSetID: <<DataSetID>>
ApplicationVersion: <<Oracle Retail Point-of-Service Version>>
StoreID: <<StoreID>>
BatchID: <<DataSetBatchID>>

#Add all the Tables Names as shown in the format below
DataFile-<<TableName>>: <<Table File Name>>
TableSequence: <<Table Names separated by comma in the order of tables to be
imported to Derby>>
```

Manifest File Example

The following is the Manifest file example for Currency DataSet:

```
DataSetName: CURRENCY
DataSetID: 5
ApplicationVersion: pos
StoreID: 04241
BatchID: 20070606084600
DataFile-CO_CNY: CO_CNY.TXT
DataFile-CO_RT_EXC: CO_RT_EXC.TXT
DataFile-CO_CNY_DNM: CO_CNY_DNM.TXT
TableSequence: CO_CNY,CO_RT_EXC,CO_CNY_DNM
```

DataSet Flat File Structure

The following is the format of the DataSet flat file:

```
<<Table Row Data with the column information separated by comma (,) and enclosed
within double quotes (") if the information is not of numeric data type. The table
row data is followed by New line character>>
```

DataSet Flat File Example

The following is the DataSet flat file example for CO_CNY table:

```
1,"US","USD","USD","US","1",2,0
2,"CA","CAD","CAD","CA","0",2,1
3,"MX","MXN","MXN","MX","0",2,3
4,"GB","GBP","GBP","GB","0",2,4
5,"EU","EUR","EUR","EU","0",2,5
6,"JP","JPY","JPY","JP","0",0,6
```

Note: All the data type values except number type must be within double quotes.

Extensibility

Extensibility is supported through the interface-based design and the use of the Spring Framework. From an extensibility stand point, an alternate implementation of any of the exposed interfaces could inherit from one of the out-of-the-box implementation classes and be injected into the system through Spring.

Additionally, the schema has been designed to enable the addition of datasets and dataset tables.

Adding New Table To Existing DataSet

Add a new row to the table CO_DT_ST_TB_IDDI and create a table script in CreateSchema.sql to add a new dataset table to the data model.

Adding More Tables To Existing DataSet Types

The following example walks through the process of adding more tables to the existing DataSet in IDDI.

1. Insert the tables to be associated with the existing DataSet by adding records to CO_DT_ST_TB_IDDI using SQL.

   Run the following queries to insert the table association to DataSet.

Example 14–1 Adding Table Association To Employee DataSet

```
insert into CO_DT_ST_TB_IDDI
(ID_DT_ST, ID_STR_RT, NM_TB, NM_FL,AI_LD_SEQ)
values
(<<Employee DataSet ID>>, <<'Store ID'>>,<<'Table1'>>,<<'Table1.txt'>>,1 );

TableName: CO_DT_ST_TB_IDDI

Column Description
ID_DT_ST : DataSet ID
ID_STR_RT: Store ID
NM_TB    : Table Name
NM_FL    : File Name of the Flat file to be generated
AI_LD_SEQ: Table Order in which the data to be exported and imported

eg: Get the Employee DataSet ID from CO_DT_ST_IDDI table

insert into CO_DT_ST_TB_IDDI
(ID_DT_ST, ID_STR_RT, NM_TB, NM_FL,AI_LD_SEQ)
values
(1,'04241','TABLE1','TABLE1.TXT',1 );

insert into CO_DT_ST_TB_IDDI
(ID_DT_ST, ID_STR_RT, NM_TB, NM_FL,AI_LD_SEQ)
values
(1,'04241','TABLE2','TABLE2.TXT',2 );
```

2. Add CREATE TABLE scripts in CreateSchema.sql.

```
CREATE TABLE "offlinedb"."TABLE1"
    ("COLUMN1" <<TYPE>> <<Constraint>>,
    "COLUMN2, <<TYPE>> <<Constraint>>)
CREATE TABLE "offlinedb"."TABLE2"
    ("COLUMN1" <<TYPE>> <<Constraint>>,
    "COLUMN2, <<TYPE>> <<Constraint>>)
```

Adding a Table to an Existing Data Set Using the Stores Build Scripts

To add a table using the build script:

1. Open <*source_directory*>\modules\utility\build.xml.
2. Find the target dataset's offline table list:

```
ordered.<data set name>.tables
```

3. Add the name of the SQL file that contains the create script.

   The create scripts are located at <*source_directory*>\modules\common\deploy\server\common\db\sql\Create.

Adding a New DataSet

To add a new DataSet:

1. Add DataSet information in CO_DT_ST_IDDI.
2. Add DataSet tables to CO_DT_ST_TB_IDDI.
3. Create <DataSetKey>Producer and <DataSetKey>Consumer classes extending from AbstractDataSetProducer and AbstractDataSetConsumer respectively.
4. Define service_config_<DataSetKey> in ServiceContext.xml.
5. Define service_<DataSetKey>Producer with class=<DataSetKey>Producer and service_<DataSetKey>Consumer wit h class=<DataSetKey>Consumer in ServiceContext.xml.
6. Add to service_<DataSetKey>Producer and service_<DataSetKey>Consumer to service_DataSetService and service_ClientDataSetService respectively in ServiceContext.xml.
7. Add DataSet key to service_FrequentProducerJob/service_InfrequentProducerJob and service_FrequentConsumerJob/service_InfrequentConsumerJob in ServiceContext.xml.
8. Add create table scripts and insert the script for the newly added DataSet in CreateSchema.sql.

Adding a New DataSet Using the Stores Build Scripts

Do the following to add a new dataset using the build script:

1. Open <*source_directory*>\modules\utility\build.xml .
2. Find the section that defines the offline table lists (target **assemble.iddi**).
3. Create the ordered list of tables, following the pattern established in the file. All create scripts are located at <*source_directory*>\modules\common\deploy\server\common\db\sql\Create.
4. Add a call to concat.file for the new data set schema, following the other calls in the file:

```
        <antcall target="concat.file">
            <param name="target.file" value="${raw.sql.file}"/>  -- The path
and name of the file being generated
            <param name="file.comment" value="-- Employee DataSet Tables"/> --
Comment added to the file ahead of the create SQL
            <param name="src.dir" value="${sql.src.dir}"/> -- Path to the
create scripts listed in the "ordered.<data set name>.tables" list
            <param name="file.list" value="${ordered.employee.tables}"/> --
Variable holding the ordered list of create scripts
            <reference refid="comment.filter" torefid="filter"/>
        </antcall>
```

Configuring Schedule For DataSet Producer And Consumer

Any existing DataSet Producer and Consumer can be individually configured to run on scheduled basis.

Configure DataSet Producer

To configure DataSet Producer:

1. Add JobDetailBean bean configuration service_<<DataSet>>ProducerJob.

```
    <bean id="service_<<DataSet>>ProducerJob"
class="org.springframework.scheduling.quartz.JobDetailBean">
        <property name="jobClass">

<value>oracle.retail.stores.foundation.iddi.DataSetProducerJob</value>
        </property>
        <property name="jobDataAsMap">
            <map>
                <entry key="producer" value-ref="service_DataSetService"/>
                <entry key="dataSets">
                    <list>
                        <ref local="service_config_<<DataSetKey>>"/>
                    </list>
                </entry>
            </map>
        </property>
    </bean>
```

Note: service_config_<<DataSetKey>> should have been configured with the DataSetKey

2. Add CronTriggerBean bean configuration service_Trigger<<DataSet>>Producer

```
    <bean id="service_Trigger<<DataSet>>Producer" class =
"org.springframework.scheduling.quartz.CronTriggerBean">
        <property name = "jobDetail">
            <ref local="service_<<DataSet>>ProducerJob"/>
        </property>
        <property name="cronExpression" value="0 0,15,30,45 0 * * ?"/>
    </bean>
```

The above DataSet is configured to run once every 15 minutes.

Note: For more information about configuring using Quartz, go to http://www.quartz-scheduler.org/documentation/quartz-2.1.x/tutorials/crontriggerl.

3. Add `service_Trigger<<DataSet>>Producer` to the SchedulerFactoryBean bean configuration:

```
    <bean id="service_ProducerSchedulerFactory"
class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
        <property name="triggers">
            <list>
                <ref local="service_TriggerFrequentProducer"/>
                <ref local="service_TriggerInfrequentProducer"/>
                <ref local="service_Trigger<<DataSet>>Producer"/>
```

```
            </list>
        </property>
    </bean>
```

Configure DataSet Consumer

To configure DataSet Consumer:

1. Add JobDetailBean bean configuration `service_<<DataSet>>ConsumerJob:`

```
    <bean id="service_<<DataSet>>ConsumerJob"
class="org.springframework.scheduling.quartz.JobDetailBean">
        <property name="jobClass">

<value>oracle.retail.stores.foundation.iddi.ClientDataSetController</value>
        </property>
        <property name="jobDataAsMap">
            <map>
               <entry key="dataSets">
                    <list>
                        <ref local="service_config_<< DataSetKey>>"/>
                    </list>
                </entry>
            </map>
        </property>
    </bean>
```

Note: `service_config_<<DataSetKey>>` should have been configured with the DataSetKey.

2. Add CronTriggerBean bean configuration `service_Trigger<<DataSet>>Consumer`:

```
    <bean id="service_Trigger<<DataSet>>Consumer" class =
"org.springframework.scheduling.quartz.CronTriggerBean">
        <property name = "jobDetail">
            <ref local="service_<<DataSet>>ConsumerJob"/>
        </property>
        <property name="cronExpression" value="0 0,15,30,45 0 * * ?"/>
    </bean>
```

   The DataSet is configured to run once every 15 minutes.

3. Add `service_Trigger<<DataSet>>Consumer` to the SchedulerFactoryBean bean configuration:

```
    <bean id=" service_clientSchedulerFactory"
class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
        <property name="triggers">
            <list>
                <ref local="service_TriggerFrequentConsumer"></ref>
                <ref local="service_TriggerInfrequentConsumer"></ref>
                <ref local="service_Trigger<<DataSet>>Consumer"/>
            </list>
        </property>
    </bean>
```

Adding New DataSet Type

The following example walks through the process of adding a new DataSet to the existing IDDI.

- Insert the new DataSet information in into the databaset table CO_DT_ST_IDDI using SQL.
- Insert the tables associated with the DataSet added to CO_DT_ST_TB_IDDI using SQL.

1. Run the following queries to insert new DataSet information and table association to DataSet.

Example 14–2 Adding New DataSet

```
insert into CO_DT_ST_IDDI
(ID_DT_ST, ID_STR_RT, NM_DT_ST)
values
(maxid+1,<<'StoreID'>> ,<<'DataSetName'>>);

TableName: CO_DT_ST_IDDI

Column Description
ID_DT_ST : DataSet ID
ID_STR_RT: Store ID
NM_DT_ST : DataSet Name

eg:
insert into CO_DT_ST_IDDI
(ID_DT_ST, ID_STR_RT, NM_DT_ST)
values
(6,'04241','NEW');
```

Example 14–3 Adding Table Association to New DataSet

```
insert into CO_DT_ST_TB_IDDI
(ID_DT_ST, ID_STR_RT, NM_TB, NM_FL,AI_LD_SEQ)
values
(<<New DataSet ID>>, <<'Store ID'>>,<<'Table1'>>,<<'Table1.txt'>>,1 );

eg:
insert into CO_DT_ST_TB_IDDI
(ID_DT_ST, ID_STR_RT, NM_TB, NM_FL,AI_LD_SEQ)
values
(6,'04241','TABLE1','TABLE1.TXT',1 );

insert into CO_DT_ST_TB_IDDI
(ID_DT_ST, ID_STR_RT, NM_TB, NM_FL,AI_LD_SEQ)
values
(6,'04241','TABLE2','TABLE2.TXT',2 );
```

2. Create `<DataSetKey>Producer` and `<DataSetKey>Consumer` classes extending from AbstractDataSetProducer and AbstractDataSetConsumer respectively.

Example 14–4 DataSetProducer Code

```
package oracle.retail.stores.domain.iddi;

import oracle.retail.stores.foundation.iddi.AbstractDataSetProducer;
import oracle.retail.stores.foundation.iddi.DataSetMetaData;
import oracle.retail.stores.foundation.iddi.TableQueryInfo;
import oracle.retail.stores.foundation.iddi.ifc.DataSetMetaDataIfc;

public class NewDataSetProducer extends AbstractDataSetProducer
{

private final String[] TABLE_FIELDS={"*"};

/**

 * NewDataSetProducer constructor
 */

public NewDataSetProducer ()
{

}
/**
* Get DataSetMetatIfc reference
*
*/
public DataSetMetaDataIfc getDataSetMetaData()
{
// Get the table names for the Key
return dataSetMetaData;
}
/**
* Initialize the MetaData for the DataSetProducer
*/
public void initializeDataSet()
{
dataSetMetaData = new DataSetMetaData(dataSetKey);
}
/**
* Create TableQueryInfo object with the column names to fetch
* @param TableName
* @return TableQueryInfo Object
*/
public TableQueryInfo getTableQueryInfo(String tableName)
{
TableQueryInfo tableQueryInfo = new TableQueryInfo(tableName);
tableQueryInfo.setTableFields(TABLE_FIELDS);
return tableQueryInfo;
}
/**
 * Finalize DataSet Method
 *
 */
public void finalizeDataSet()
{

}
}
```

Example 14–5 DataSetConsumer Code

```
package oracle.retail.stores.domain.iddi;

import oracle.retail.stores.foundation.iddi.AbstractDataSetConsumer;

//------------------------------------------------------------------------
/**
    The NewDataSetConsumer defines methods that the
    application calls to import Employee dataset files into
    offline database.
     @version $Revision: $
**/
//------------------------------------------------------------------------

public class NewDataSetConsumer extends AbstractDataSetConsumer
{         /**  DataSet key name for currency dataset.

    */    private String dataSetKey = null;

   // ------------------------------------------------------------------------
    /**
        @return Returns the dataSetKey
    **/
    //------------------------------------------------------------------------

    public String getDataSetKey()
    {

return dataSetKey;
    }

    //  ------------------------------------------------------------------------
    /**
      @param dataSetKey The DataSetKey to set
    **/
    //------------------------------------------------------------------------

    public void setDataSetKey(String dataSetKey)
    {

this.dataSetKey = dataSetKey;

  }
}
```

3. Define `service_config_<<DataSetKey>>` in ServiceContext.xml:

```
<bean id="service_config_<<datasetKey>> " class="java.lang.String">
    <constructor-arg type="java.lang.String" value="<<DataSetKey>>"/>
</bean>eg:    <bean id="service_config_NEW_KEY" class="java.lang.String">
    <constructor-arg type="java.lang.String" value="NEW"/>
</bean>
```

4. Define `service_<<DataSetKey>>Producer` with `class=<DataSetKey>Producer` and `service_<<DataSetKey>>Consumer` with `class=<DataSetKey>Consumer` in ServiceContext.xml:

```
<bean id="service_NewProducer"
class="oracle.retail.stores.domain.iddi.NewDataSetProducer" lazy-init="true"
singleton=”true”>
```

```
        <property name="dataSetKey" ref="service_config_NEW_KEY"/>
        <property name="dataExportFilePath" ref="service_config_
DataExportFilePath"/>
        <property name="dataExportZipFilePath" ref="service_config_
DataExportZipFilePath"/>
    </bean>
    <bean id="service_NewConsumer"
 class="oracle.retail.stores.domain.iddi.NewDataSetConsumer"
            lazy-init="true"
            singleton="true">
        <property name="dataSetKey" ref="service_config_NEW_KEY"/>
        <property name="dataImportFilePath" ref="service_config_
DataImportFilePath"/>
    </bean>
```

5. Add to `service_<<DataSetKey>>Producer` and `service_<<DataSetKey>>Consumer` to `service_DataSetService` and `service_ClientDataSetService` respectively in ServiceContext.xml

```
   <bean id="service_DataSetService"
class="oracle.retail.stores.foundation.iddi.DataSetService" singleton="true">
        <property name="producers">
            <map>
                <entry key-ref="service_config_EMP_KEY" value-ref="service_
EmployeeProducer"/>
                <entry key-ref="service_config_ITM_KEY" value-ref="service_
ItemProducer"/>
                <entry key-ref="service_config_PRC_KEY" value-ref="service_
AdvancedPricingProducer"/>
                <entry key-ref="service_config_TAX_KEY" value-ref="service_
TaxProducer"/>
                <entry key-ref="service_config_CUR_KEY" value-ref="service_
CurrencyProducer"/>
                <entry key-ref="service_config_NEW_KEY" value-ref="service_
NewProducer"/>
            </map>
        </property>
    </bean>
    <bean id="service_ClientDataSetService"
     class="oracle.retail.stores.foundation.iddi.ClientDataSetService"
singleton="true">
        <property name="consumers">
            <map>
                <entry key-ref="service_config_EMP_KEY" value-ref="service_
EmployeeConsumer"/>
                <entry key-ref="service_config_CUR_KEY" value-ref="service_
CurrencyConsumer"/>
                <entry key-ref="service_config_TAX_KEY" value-ref="service_
TaxConsumer"/>
                <entry key-ref="service_config_ITM_KEY" value-ref="service_
ItemConsumer"/>
                <entry key-ref="service_config_PRC_KEY" value-ref="service_
AdvancedPricingConsumer"/>
                <entry key-ref="service_config_NEW_KEY" value-ref="service_
NewConsumer"/>
            </map>
        </property>
        <property name="dataImportFilePath" ref="service_config_
DataImportFilePath"/>
    </bean>
```

6. Add DataSet key to `service_FrequentProducerJob/service_InfrequentProducerJob` and `service_FrequentConsumerJob/service_InfrequentConsumerJob` in ServiceContext.xml

```
    <bean id="service_FrequentProducerJob"
class="org.springframework.scheduling.quartz.JobDetailBean">
        <property name="jobClass">

<value>oracle.retail.stores.foundation.iddi.DataSetProducerJob</value>
        </property>
        <property name="jobDataAsMap">
            <map>
                <entry key="producer" value-ref="service_DataSetService"/>
                <entry key="dataSets">
                    <list>
                        <ref local="service_config_EMP_KEY"/>
                        <ref local="service_config_PRC_KEY"/>
                        <ref local="service_config_TAX_KEY"/>
                        <ref local="service_config_NEW_KEY"/>
                    </list>
                </entry>
            </map>
        </property>
    </bean>

<bean id="service_FrequentConsumerJob"
class="org.springframework.scheduling.quartz.JobDetailBean">
        <property name="jobClass">

<value>oracle.retail.stores.foundation.iddi.ClientDataSetController</value>
        </property>
        <property name="jobDataAsMap">
            <map>
               <entry key="dataSets">
                    <list>
                        <ref local="service_config_EMP_KEY"/>
                        <ref local="service_config_PRC_KEY"/>
                        <ref local="service_config_TAX_KEY"/>
                        <ref local="service_config_NEW_KEY"/>
                    </list>
                </entry>
            </map>
        </property>
    </bean>
```

7. Add CREATE TABLE scripts and insert scripts to newly added DataSet in CreateSchema.sql.

```
CREATE TABLE "offlinedb"."TABLE1"
    ("COLUMN1" <<TYPE>> <<Constraint>>,
 "COLUMN2, <<TYPE>> <<Constraint>>)
CREATE TABLE "offlinedb"."TABLE2"
    ("COLUMN1" <<TYPE>> <<Constraint>>,
 "COLUMN2, <<TYPE>> <<Constraint>>)
insert into CO_DT_ST_IDDI(ID_DT_ST, ID_STR_RT, NM_DT_ST)
values(6,'04241','NEW');
```

Adding a New DataSet Type Using the Stores Build Scripts

Do the following to add a new dataset type using the build script:

1. Open <*source_directory*>\modules\utility\build.xml .
2. Find the section that defines the offline table lists (target **assemble.iddi**).
3. Create the ordered list of tables, following the pattern established in the file. All create scripts are located at <*source_directory*>\modules\common\deploy\server\common\db\sql\Create.
4. Add a call to concat.file for the new data set schema, following the other calls in the file:

```
        <antcall target="concat.file">
            <param name="target.file" value="${raw.sql.file}"/>  -- The path
and name of the file being generated
            <param name="file.comment" value="-- Employee DataSet Tables"/> --
Comment added to the file ahead of the create SQL
            <param name="src.dir" value="${sql.src.dir}"/> -- Path to the
create scripts listed in the "ordered.<data set name>.tables" list
            <param name="file.list" value="${ordered.employee.tables}"/> --
Variable holding the ordered list of create scripts
            <reference refid="comment.filter" torefid="filter"/>
        </antcall>
```

Changing Point-of-Service Client Database Vendor

The Point-of-Service client uses the Derby database. However, the modifications to the code are minimal for replacing the Point-of-Service client database from Derby to another database. Do the following to change the Point-of-Service client database:

1. Add `Offline<<DBName>>Helper` class which implements `offlineDBHelperIfc`.
2. Change the installer to have new database driver jar file paths.
3. Update the `"<POOL name="jdbcpool class="DataConnectionPool" package="oracle.retail.stores.foundation.manager.data">"` section of PosLFFDataTechnician.xml file with the driver, databaseUrl, userid, password.

15

Audit Logging

The audit log retains events that are logged to the file system. Audit Logs include access, search, view (generate), print and export for the following functional areas in Point-of-Service:

- Daily Operations Audit Log Events
- Employee Audit Log Events
- Login, Logout, Lockout Audit Log Events
- Password Audit Log Events
- Point-of-Service Transaction Events
- Role Audit Log Events
- Till Audit Log Events
- Parameter Log Events

Each event has a specific set of components that must be present in the Audit Log. Each event is required to have an event name, event status, system date and system time in which the event was completed. The status of an event can either be Success or Failure. If an event was executed without interruption and the data of the event is saved to persistent storage, the events status is **Success**. If a database exception occurs after the operator or system has finished the event, the events status is **Failure**. If any exception occurs before the activity is saved or if the operator selects to leave the application, no event is logged.

The Audit Log is implemented using a log4j logging infrastructure.

Log4j is an Apache (`www.apache.org`) utility used to assist applications in meaningful logging. These log statements are printed in a format that can be used for further processing, such as reporting.

The log4j mechanism works on properties/XML configuration files where the minimum logging level for the application is mentioned. Throughout the application, where a statement must be logged, the Log4j API for a particular level is called. If the application LOG4J is setup for a level that is equal to or lower in priority to the API being invoked, then that statement is logged; if the application LOG4J is setup for a level that is not equal to or lower in priority to the API being invoked, then that statement is not logged. Therefore, if the configuration is for a WARN level, then INFO, DEBUG and TRACE statements are not logged.

The following are the various logging levels available, in increasing order of priority:

- TRACE
- DEBUG

- INFO
- WARN
- ERROR
- FATAL

If the application logging level is set at WARN, and in the application the INFO API is being called to log, that statement will not get logged as WARN is a higher priority than INFO. All log statements which are WARN level or higher only are logged.

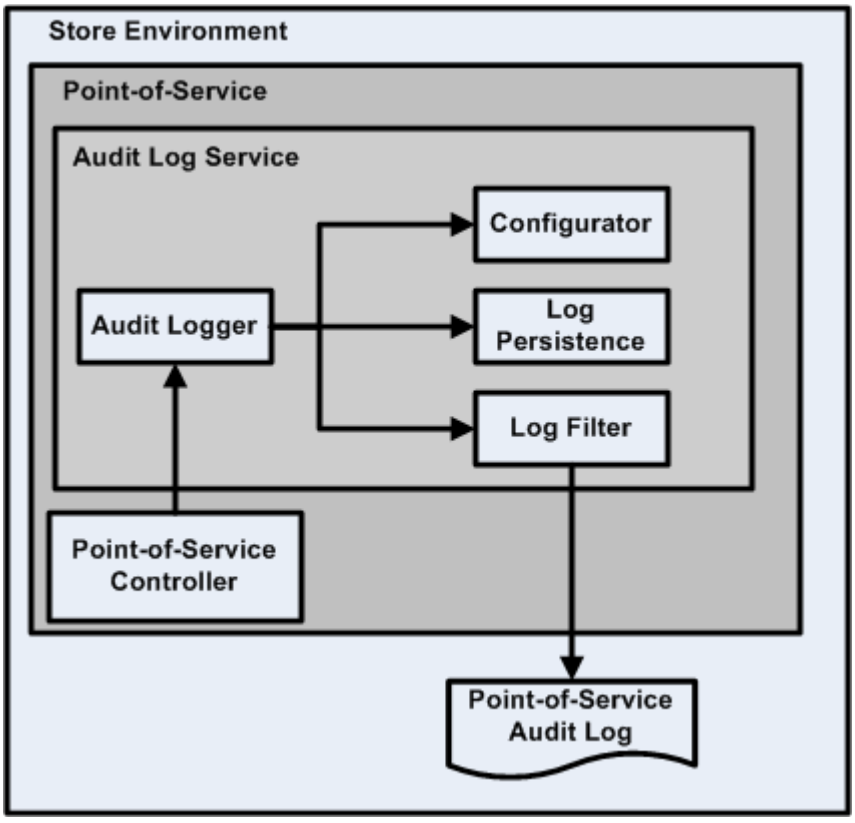
The best practice is to have the application logging level set at INFO for production systems.

The Audit Log uses the LOG4J system to log the audit statements. The audit log code is written such that it invokes the INFO API to log the statements.

Therefore, if the log4j configuration for Audit Logging is set to INFO or DEBUG then the application will log the audit statements. If set to anything higher than INFO, no audit statements will be logged.

Figure 15–1 is a Point-of-Service common configuration for the Audit Logging subsystem:

Figure 15–1 Audit Log in Point-of-Service



Configuring the Audit Log

> **Note:** The *Oracle Retail POS Suite Security Guide* describes specific security features and implementation guidelines for the POS Suite products.

You can configure the Audit Log using configuration files. To update the logging infrastructure, update the Spring `ServiceContext.xml` file to point the various infrastructure bean IDs to any alternate implementation classes you want to provide.

```
Bean ID: service_AuditLogger
Class: oracle.retail.stores.commerceservices.audit.AuditLoggerService
```

Because the Audit Log is using Log4J as the underlying logging mechanism, you can also control the logging layout, location, and content by updating the `log4j.xml` file.

- All log events will be logged at the INFO level, so to disable logging entirely, change the log level to WARN or above for the event package path.
- Additionally, each logging event is represented in the `log4j.xml` file through the event's package path, so to filter a specific event, just update that event's level to WARN or above.
- As with all Log4J deployments, updating the layout of the log events or their location is a matter of setting the layout in the configuration file and updating the appender to point to a different file name. Another option is to use an entirely different appender to write to a database or even a JMS queue.

Internationalize Static Text/Date/Time/Currency

Use AuditLoggerI18NHelper, which has the following methods:

- getString(String key)
- getFormattedDate(Date)
- getFormattedTime(Date)
- getFormattedCurrency(String)

All these methods will return the data in the application's default locale.

> **Note:** Before setting Auditlog event objects to log database data, retrieve the database data in the client's default locale by calling get<*FieldName*>(Locale) method of domain classes.

The following is an example of settings that might be used in a log4j.xml file:

Example 15–1 Audit Log Configuration Changes in the log4j.xml File

```
<!-- AUDIT Logging -->

  <category name="oracle.retail.stores.commerceservices.audit.event">
      <!-- The following elements are commented to prevent duplicate logging
          <priority value="INFO" />
          <appender-ref ref="AUDIT"/>
      -->
  </category>
```

```
    <category
name="log4j.additivity.oracle.retail.stores.commerceservices.audit.event=false">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.ENTER_
BUSINESS_DATE">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.START_OF_
DAY">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.END_OF_DAY">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.BANK_
DEPOSIT">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.BANK_
DEPOSIT_REPORT_EXPORTED">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.REGISTER_
OPEN">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.REGISTER_
CLOSE">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.TILL_
RECONCILE">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.TILL_OPEN">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.TILL_CLOSE">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.TILL_
SUSPEND">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.TILL_
RETRIEVE">
            <priority value="INFO" />
```

```
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.COUNT_FLOAT_
AT_RECONCILE">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.ADD_
EMPLOYEE">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.MODIFY_
EMPLOYEE_INFORMATION">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.ADD_
TEMPORARY_EMPLOYEE">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.MODIFY_
TEMPORARY_EMPLOYEE_INFORMATION">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.RESET_
EMPLOYEE_PASSWORD">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.RESET_
TEMPORARY_EMPLOYEE_PASSWORD">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.CHANGE_
PASSWORD">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.USER_
LOGOUT">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.USER_LOGIN">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.USER_LOCK_
OUT">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.ADD_ROLE">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
```

```
    <category name="oracle.retail.stores.commerceservices.audit.event.ADD_USER">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.EDIT_ROLE">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.REMOVE_
ROLE">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.MODIFY_
APPLICATION_PARAMETER">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.MODIFY_
PARAMETER_IN_LIST">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.ADD_
PARAMETER_LIST_FOR_DISTRIBUTION">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.REMOVE_
PARAMETER_LIST">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.DISTRIBUTE_
PARAMETER_LIST">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.TRANSACTION_
TENDERED_WITH_CREDIT_CARD">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
    <category name="oracle.retail.stores.commerceservices.audit.event.TRANSACTION_
TENDERED_WITH_DEBIT_CARD">
            <priority value="INFO" />
            <appender-ref ref="AUDIT"/>
    </category>
```

Daily Operations Audit Log Events

The following are daily operations audit log events.

Enter Business Date

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Enter Business Date system setting equals INFO.

- Event data collection starts when the operator enters a business date.
- Event data collection ends when the operator selects **Next**.
- There is no failure condition to this event.

Table 15–1 Enter Business Date Event Components

Event Components	Notes
Event Name	Enter Business Date.
Event Status	Success.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
Business Date	Entered Business date.
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register Number	For Point-of-Service, the register number at which the event transpired.
Till ID	Till ID at which the event transpired.

Start of Day

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Start of Day system setting equals INFO.

- Event data collection starts when the operator selects to execute Start of Day functionality.
- Event data collection ends when the system displays that the store is opened.
- The format of this event is dependent on the Count Operating Fund at Start of Day parameter setting.
- Failure can happen only when there is some technical error.

Table 15–2 Start of Day Event Components

Event Components	Notes
Event Name	Start of Day.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).

Table 15–2 (Cont.) Start of Day Event Components

Event Components	Notes
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register Number	For Point-of-Service, the register number at which the event transpired.
Operating Fund Amount	■ Entered cash amount for Count Operating Fund at Start of Day equals Summary. ■ Total of entered cash amount for Count Operating Fund at Start of Day equals Detail. ■ Equal to the Operating Fund Expected Amount when Count Operating Fund at Start of Day equals No.
Pennies	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day equals Detail.
Nickels	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day equals Detail.
Dimes	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day equals Detail.
Quarters	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day equals Detail.
Half-Dollars	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day equals Detail.
$1 Coins	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day equals Detail.
$2 Coins	Entered currency denomination amount. Only recorded if a value is entered, Count Operating Fund at Start of Day equals Detail and Canadian currency is the base currency.
$1 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day equals Detail.
$2 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day equals Detail.
$5 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day equals Detail.
$10 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day equals Detail.
$20 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day equals Detail.
$50 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day equals Detail.
$100 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day equals Detail.
Store Status	■ Open. ■ Close.

End of Day

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the End of Day system setting equals INFO.

- Event data collection starts when the operator selects to begin end of day.
- Event data collection ends when the system assigns a transaction number.
- The format of this event is dependent on the Count Operating Fund at End of Day parameter setting.
- Event failure can happen only due to technical reasons, for example, unable to get next sequence number for transaction, transaction creation exception, EJB call exception or if the financial totals are not found in the database.

Table 15–3 End of Day Event Components

Event Components	Notes
Event Name	End of Day.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register Number	For Point-of-Service, the register number at which the event transpired.
Operating Fund Amount	■ Entered cash amount for Count Operating Fund at End of Day equals Summary. ■ Total of entered cash amount for Count Operating Fund at End of Day equals Detail. ■ Equal to the Operating Fund Expected Amount when Count Operating Fund at End of Day equals No.
Pennies	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day equals Detail.
Nickels	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day equals Detail.
Dimes	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day equals Detail.
Quarters	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day equals Detail.
Half-Dollars	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day equals Detail.
$1 Coins	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day equals Detail.
$1 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day equals Detail.

Table 15–3 (Cont.) End of Day Event Components

Event Components	Notes
$2 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day equals Detail.
$5 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day equals Detail.
$10 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day equals Detail.
$20 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day equals Detail.
$50 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day equals Detail.
$100 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day equals Detail.
Transaction Number	Transaction number assigned by the system to the store close event.

Register Open

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Register Open system setting equals INFO.

- Event data collection starts when the operator selects to open a register.
- Event data collection ends when the system assigns a transaction number.
- If more than one register is selected to open at one time, a separate independent event is written to the audit log. Each opened register is assigned an individual transaction number.
- Event failure can happen only due to technical reasons, such as unable to get next sequence number for transaction, transaction creation exception, EJB call exception or if the financial totals are not found in the database.

Table 15–4 Register Open Event Components

Event Components	Notes
Event Name	Register Open.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register Number	For Point-of-Service, the register number at which the event transpired.
Transaction Number	Transaction number assigned by the system to the opened register.

Register Close

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Register Close system setting equals INFO.

- Event data collection starts when the operator selects to close a register.
- Event data collection ends when the system assigns a transaction number.
- Event failure can happen only due to technical reasons, such as unable to get next sequence number for transaction, transaction creation exception, EJB call exception or if the financial totals are not found in the database.

Table 15–5 Register Close Event Components

Event Components	Notes
Event Name	Register Close.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register Number	For Point-of-Service, the register number at which the event transpired.
Transaction Number	Transaction number assigned by the system to the closed register.

Point-of-Service Transaction Events

The following are Point-of-Service transaction events.

Transaction Tendered with Credit Card

This is a Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Transaction Tendered with Credit Card system setting equals INFO.

- Event data collection starts when the operator selects Credit/Debit from Tender Options menu and has entered the card number.
- The operator has chosen Credit from the Tender Options menu and has entered the card number.
- Event data collection ends when a credit card tender has been added to the transaction with the authorization status pending.
- Failure Condition will be logged only in case of technical failures such as Database is down.

Table 15–6 Transaction Tendered with Credit Card Event Components

Event Components	Notes
Event Name	Transaction Tendered with Credit Card.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register Number	The register number at which the event transpired.
Till ID	The Till ID at which the event transpired.
Card type	The card type presented at time of tender.
Card number	■ The card number presented at the time of tender. ■ Only display last 4 digits of card number. For example: `xxxx xxxx xxxx 1111`
Amount	The amount the card is being charged at the time of tender.
Entry method (manual/auto)	The method used to enter the card. Operator input on keyboard is manual and a scan, or swipe on the device or keyboard is auto.
MAG stripe (if swiped)	An indicator if swiped on the MSR.
Authorization Status(Pending)	The status of authorization is pending until a response is returned.

Transaction Tendered with Debit Card

This is a Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Transaction Tendered with Debit Card system setting equals INFO.

- Event data collection starts when the operator selects Credit/Debit from the Tender Options menu.
- The operator has chosen Debit from the Tender Options menu.
- Event data collection ends when a debit card tender has been added to the transaction with the authorization status pending.
- Failure Condition is logged only in case of technical failures such as Database is down.

Table 15–7 Transaction Tendered with Debit Card Event Components

Event Components	Notes
Event Name	Sale Transaction Tendered with Debit Card.
Event Status	■ SUCCESS. ■ FAILURE.
Event Originator	Class Name and Method Name (`ClassName.methodName`).
System Date	System date of the event.

Table 15–7 (Cont.) Transaction Tendered with Debit Card Event Components

Event Components	Notes
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register Number	The register number at which the event transpired.
Till ID	The Till ID at which the event transpired.
Debit type	The card type presented at time of tender.
Card number	The card number presented at the time of tender. Only display last 4 digits of card number. For example: `xxxx xxxx xxxx 1111`
Amount	The amount the card is being charged at the time of tender.
Entry method (manual/auto)	The method used to enter the card. Operator input on keyboard is manual and a scan, or swipe on the device or keyboard is auto.
MAG stripe (if swiped)	An indicator if swiped on the MSR.
Authorization Status(Pending)	The status of authorization is pending until a response is returned.

Employee Audit Log Events

The following are employee audit log events.

Modify Employee Information

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Modify Employee Information system setting equals INFO.

- Event data collection starts when the operator edits an employees information.
- Event data collection ends when the operator selects Save.
- If the operator selects Save but has not modified any employee information the event name is Modify Employee Information
- Employee getting modified is not found in the Database is the only failure condition possible.

Table 15–8 Modify Employee Information Event Components

Event Components	Notes
Event Name	Modify Employee Information.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number at which the event transpired.

Table 15–8 (Cont.) Modify Employee Information Event Components

Event Components	Notes
User ID	User ID performing the event.
Register Number	For Point-of-Service, the register number at which the event transpired.
Employee ID	Employee ID of the modified employee.

Modify Temporary Employee Information

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Modify Temporary Employee Information system setting equals INFO.

- Event data collection starts when the operator edits a temporary employees information.
- Event data collection ends when the operator selects Save.
- If the operator selects Save but has not modified any temporary employee information the event name is Modify Employee Information
- Employee getting modified is not found in the Database is the only failure condition possible.

Table 15–9 Modify Temporary Employee Information Event Components

Event Components	Notes
Event Name	Modify Temporary Employee Information.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register Number	For Point-of-Service, the register number at which the event transpired.
Employee ID	Employee ID of the modified temporary employee.

Add Employee

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Add Employee system setting equals INFO.

- Event data collection starts when the operator selects to add an employee.
- Event data collection ends when the operator selects Save.
- Failure Event is when the login ID provided is already in use.

Table 15–10 Add Employee Event Components

Event Components	Notes
Event Name	Add Employee.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register Number	For Point-of-Service, the register number at which the event transpired.
Employee ID	Employee ID.
First Name	Entered first name.
Middle Name	Entered middle name.
Last Name	Entered last name.
Employee Login ID	Entered login ID.
Role Name	Selected role.
Employee Status	Selected employee status.

Add Temporary Employee

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Add Temporary Employee system setting equals INFO.

- Event data collection starts when the operator selects to add a temporary employee.
- Event data collection ends when the operator selects Save.
- Failure Event is when the login ID provided is already in use.

Table 15–11 Add Temporary Employee Event Components

Event Components	Notes
Event Name	Add Temporary Employee.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register Number	For Point-of-Service, the register number at which the event transpired.
Employee ID	Employee ID.

Table 15–11 (Cont.) Add Temporary Employee Event Components

Event Components	Notes
First Name	Entered first name.
Middle Name	Entered middle name.
Last Name	Entered last name.
Employee Login ID	Entered login ID.
Role Name	Selected role.
Store#	Entered store number.
Days Valid	Selected remaining days valid.
Employee Status	Selected employee status.

Login, Logout, Lockout Audit Log Events

The following are login, logout and lockout audit log events.

User Login

This is a Back Office, Point-of-Service and Central Office event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the User Login system setting equals INFO.

- Event data collection starts when the operator enters their login information.
- Event data collection ends when the operator selects to log in.
- Even failure can happen only when there is a technical exception.

Table 15–12 User Login Event Components

Event Components	Notes
Event Name	User Login.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at. Only applicable for Back Office.
User ID	User ID is recorded.
Register Number	For Point-of-Service, the register number at which the event transpired.

User Lock Out

This is a Back Office, Point-of-Service and Central Office event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the User Lock Out system setting equals INFO.

- Event data collection starts and ends when the user attempts to log in and is locked out due to unsuccessful login attempts or an expired password.
- No failure condition.

Table 15–13 User Lock Out Event Components

Event Components	Notes
Event Name	User Lock Out.
Event Status	Success.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at. Only applicable for Back Office.
User ID	User ID is recorded.
Register Number	For Point-of-Service, the register number at which the event transpired.
Lockout Reason	■ <ARG> consecutive unsuccessful login attempts. (<ARG> equals Number of login attempts). ■ Expired Password.

User Logout

This is a Back Office, Point-of-Service and Central Office event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the User Logout system setting equals INFO.

- Event data collection starts and ends when the user selects to log out.
- No Failure Condition.

Table 15–14 User Logout Event Components

Event Components	Notes
Event Name	User Logout.
Event Status	Success.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at. Applicable only for Back Office.
User ID	User ID is recorded.
Register Number	For Point-of-Service, the register number at which the event transpired.

Password Audit Log Events

The following are password audit log events.

Change Password

This is a Back Office, Point-of-Service and Central Office event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Change Password system setting equals INFO.

- Event data collection starts when the operator selects or is prompted to change their password.

- Event data collection ends when the operator selects to save their new password.
- Failure Condition will occur when the employee or user for whom the password is being changed does not exist in the database. If the new password does not meet the password criteria, a failure is also logged.

Table 15–15 Change Password Event Components

Event Components	Notes
Event Name	User Change Password.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
System Time	Time of the event.
Store ID	For Back Office, the store number at which the event transpired.
User ID	User ID is recorded.
Register Number	For Point-of-Service, the register number at which the event transpired.

Reset Employee Password

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Reset Employee Password system setting equals INFO.

- Event data collection starts when the operator selects to reset an employee's password.
- Event data collection ends when the operator selects Yes.
- Failure Condition will be logged only in case of technical failures such as the database is down.

Table 15–16 Reset Employee Password Event Components

Event Components	Notes
Event Name	Reset Employee Password.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register Number	For Point-of-Service, the register number at which the event transpired.
Employee ID	Employee ID whose password was reset.

Reset Temporary Employee Password

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Reset Temporary Employee Password system setting equals INFO.

- Event data collection starts when the operator selects to reset an employees password.
- Event data collection ends when the operator selects Yes.
- Failure Condition will be logged only in case of technical failures such as DB is down.

Table 15–17 Reset Temporary Employee Password Event Components

Event Components	Notes
Event Name	Reset Temporary Employee Password.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register Number	For Point-of-Service, the register number at which the event transpired.
Employee ID	Employee ID whose password was reset.

Role Audit Log Events

The following are role audit log events.

Edit Role

This is a Back Office, Point-of-Service and Central Office event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Edit Role system setting equals INFO.

- Event data collection starts when the operator edits the role.
- Event data collection ends when the operator selects Save.
- Failure Condition only due to Technical exceptions.

Table 15–18 Edit Role Event Components

Event Components	Notes
Event Name	Edit Role.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.

Table 15–18 (Cont.) Edit Role Event Components

Event Components	Notes
System Time	Time of the event.
User ID	User ID performing the event.
Register Number	For Point-of-Service, the register number at which the event transpired.
Role Name	Selected role name.

Add Role

This is a Back Office, Point-of-Service and Central Office event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Add Role system setting equals INFO.

- Event data collection starts when the operator selects Add.
- Event data collection ends when the operator selects to save the role settings for the role.
- Failure Condition only due to technical exceptions

Table 15–19 Add Role Event Components

Event Components	Notes
Event Name	Add Role.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
System Time	Time of the event.
User ID	User ID performing the event.
Register Number	For Point-of-Service, the register number at which the event transpired.
Role Name	Entered role name.
Role Setting	Selected role setting, includes application full name and feature.

Till Audit Log Events

The following are till audit log events.

Till Open

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Till Open system setting equals INFO.

- Event data collection starts when the operator selects to open a till.
- Event data collection ends when the system assigns a transaction number.
- The format of this event is dependent on the Count Float at Open parameter setting.

- Event failure can happen only due to technical reasons, such as unable to get next sequence number for transaction, transaction creation exception, EJB call exception or if the financial totals are not found in the database.

Table 15–20 Till Open Event Components

Event Components	Notes
Event Name	Till Open.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register ID	Complete Register ID value is recorded.
Till ID	Complete Till ID value is recorded.
Operator ID	Operator ID is the user assigned to the till, not the logged-in user ID.
Float Amount	■ Entered amount when Count Float at Open equals Summary. ■ Total amount all denominations entered when Count Float at Open equals Detail. ■ Equal to the Float Amount when Count Float at Open equals No.
Pennies	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open equals Detail.
Nickels	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open equals Detail.
Dimes	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open equals Detail.
Quarters	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open equals Detail.
Half-Dollars	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open equals Detail.
$1 Coins	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open equals Detail.
$1 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open equals Detail.
$2 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open equals Detail.
$5 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open equals Detail.
$10 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open equals Detail.
$20 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open equals Detail.

Table 15–20 (Cont.) Till Open Event Components

Event Components	Notes
$50 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open equals Detail.
$100 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open equals Detail.
Transaction Number	Transaction number assigned to closed register.

Till Suspend

This is a Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Till Suspend system setting equals INFO.

- Event data collection starts when the operator selects to suspend a till.
- Event data collection ends when the system assigns a transaction number.
- Event failure can happen only due to technical reasons, such as unable to get next sequence number for transaction, transaction creation exception, EJB call exception or if the financial totals are not found in the database.

Table 15–21 Till Suspend Event Component

Event Components	Notes
Event Name	Till Suspend.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (`ClassName.methodName`).
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register Number	The register number at which the event transpired.
Till ID	The Till ID at which the event transpired.
Operator ID	Operator ID is user assigned to the till not the logged in user ID.

Till Resume

This is a Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Till Resume system setting equals INFO.

- Event data collection starts when the operator selects to retrieve a suspended till.
- Event data collection ends when the system assigns a transaction number.
- Event failure can happen only due to technical reasons, such as unable to get next sequence number for transaction, transaction creation exception, EJB call exception or if the financial totals are not found in the database.

Table 15–22 Till Resume Event Component

Event Components	Notes
Event Name	Till Resume.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (`ClassName.methodName`).
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register Number	The register number at which the event transpired.
Till ID	The Till ID at which the event transpired.
Operator ID	Operator ID is user assigned to the till not the logged in user ID.

Till Close

This is a Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Till Close system setting equals INFO.

- Event data collection starts when the operator selects to close a till.
- Event data collection ends when the system assigns a transaction number.
- Event failure can happen only due to technical reasons, for example, unable to get next sequence number for transaction, transaction creation exception, EJB call exception or if the financial totals are not found in the database.

Table 15–23 Till Close Event Component

Event Components	Notes
Event Name	Till Close.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register Number	The register number at which the event transpired.
Till ID	The Till ID at which the event transpired.
Operator ID	Operator ID is the user assigned to the till, not the logged-in user ID.

Count Float at Reconcile

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Count Float at Reconcile system setting equals INFO.

- Event data collection starts when the system checks the Count Float at Reconcile parameter.
- Event data collection ends when the count float amount has been entered or accepted.
- The format of this event is dependent on the Count Float at Reconcile parameter setting.
- Event failure can happen only due to technical reasons, such as unable to get next sequence number for transaction, transaction creation exception, EJB call exception or if the financial totals are not found in the database.

Table 15–24 Count Float at Reconcile Event Components

Event Components	Notes
Event Name	Count Float at Reconcile.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.
Register ID	Complete Register ID value is recorded.
Till ID	Complete Till ID value is recorded.
Operator ID	Operator ID is user assigned to the till not the logged in user ID.
Float Amount	■ Entered amount when Count Float at Reconcile equals Summary. ■ Total amount all denominations entered when Count Float at Reconcile equals Detail. ■ Equal to the Float Amount when Count Float at Reconcile equals No.
Pennies	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile equals Detail.
Nickels	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile equals Detail.
Dimes	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile equals Detail.
Quarters	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile equals Detail.
Half-Dollars	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile equals Detail.
$1 Coins	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile equals Detail.

Table 15–24 (Cont.) Count Float at Reconcile Event Components

Event Components	Notes
$1 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile equals Detail.
$2 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile equals Detail.
$5 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile equals Detail.
$10 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile equals Detail.
$20 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile equals Detail.
$50 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile equals Detail.
$100 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile equals Detail.

Till Reconcile

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Till Reconcile system setting equals INFO.

- Event data collection starts when the system checks the Count Till at Reconcile parameter.
- If the Count Till at Reconcile equals No, event data collection ends when the system assigns a transaction number.
- If the Count Till at Reconcile equals Detail or Summary, event data ends when the system displays the Reconcile Till Count Report.
- The format of this event is dependent on the Count Till at Reconcile parameter setting and Blind Close parameter setting.
- Event failure can happen only due to technical reasons, such as unable to get next sequence number for transaction, transaction creation exception, EJB call exception or if the financial totals are not found in the database.

Table 15–25 Till Reconcile Event Components

Event Components	Notes
Event Name	Till Reconcile.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
System Time	Time of the event.
Business Date	Business date of the event.
Store ID	Store number at which the event transpired.
User ID	User ID performing the event.

Table 15–25 (Cont.) Till Reconcile Event Components

Event Components	Notes
Register ID	Complete Register ID value is recorded.
Till ID	Complete Till ID value is recorded.
Operator ID	Operator ID is user assigned to the till not the logged in user ID.
Cash Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered currencies Count Till at Reconcile equals Detail and the currency was received.
Pennies	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile equals Detail.
Nickels	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile equals Detail.
Dimes	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile equals Detail.
Quarters	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile equals Detail.
Half-Dollars	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile equals Detail.
$1 Coins	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile equals Detail.
$1 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile equals Detail.
$2 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile equals Detail.
$5 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile equals Detail.
$10 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile equals Detail.
$20 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile equals Detail.
$50 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile equals Detail.
$100 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile equals Detail.
Check Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total amount all deposited checks entered when Count Till at Reconcile equals Detail.
<ARG> Check	■ Check amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each check entered. ■ <ARG> equals the number of the Check.
Credit Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Credit when Count Till at Reconcile equals Detail and the tender was received.

Table 15–25 (Cont.) Till Reconcile Event Components

Event Components	Notes
<ARG> Credit	■ Credit amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each credit entered. ■ <ARG> equals the number of the Credit.
Debit Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Debit when Count Till at Reconcile equals Detail and the tender was received.
<ARG> Debit	■ Debit amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each Debit entered. ■ <ARG> equals the number of the Debit.
Gift Card Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Gift Card when Count Till at Reconcile equals Detail and the tender was received.
<ARG> Gift Card	■ Gift Card amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each Gift Card entered. ■ <ARG> equals the number of the Gift Card.
Gift Certificate Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Gift Certificate when Count Till at Reconcile equals Detail and the tender was received.
<ARG> Gift Certificate	Gift Certificate amount entered when Count Till at Reconcile equals Detail. There is an audit log entry for each Gift Certificate entered.
Travelers Check Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Travelers Check when Count Till at Reconcile equals Detail and the tender was received.
<ARG> Travelers Check	■ Travelers Check amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each Travelers Check entered. ■ <ARG> equals the number of the Travelers Check.
Coupon Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Coupon when Count Till at Reconcile equals Detail and the tender was received.
<ARG> Coupon	■ Coupon amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each Coupon entered. ■ <ARG> equals the number of the Coupon.

Table 15–25 (Cont.) Till Reconcile Event Components

Event Components	Notes
Store Credit Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Store Credit when Count Till at Reconcile equals Detail and the tender was received.
<ARG> Store Credit	■ Store Credit amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each Store Credit entered. ■ <ARG> equals the number of the Store Credit.
Mall Certificate Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Mall Certificate when Count Till at Reconcile equals Detail and the tender was received.
<ARG> Mall Certificate	■ Mall Certificate amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each Mall Certificate entered. ■ <ARG> equals the number of the Mall Certificate.
Purchase Order Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Purchase Order when Count Till at Reconcile equals Detail and the tender was received.
<ARG> Purchase Order	■ Purchase Order amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each Purchase Order entered. ■ <ARG> equals the number of the Purchase Order.
E-Check Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered E-Check when Count Till at Reconcile equals Detail and the tender was received.
<ARG> E-Check	■ E-Check amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each E-Check entered. ■ <ARG> equals the number of the E-check.
Canadian Cash Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered currencies when Count Till at Reconcile equals Detail and the currency was received.
$2 Coins	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile equals Detail.
Canadian Check Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Canadian Check when Count Till at Reconcile equals Detail and the tender was received.

Table 15–25 (Cont.) Till Reconcile Event Components

Event Components	Notes
<ARG> Canadian Check	■ Canadian Check amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each Canadian Check entered. ■ <ARG> equals the number of the Canadian Check.
Canadian Travelers Check Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Canadian Travelers Check when Count Till at Reconcile equals Detail and the tender was received.
<ARG> Canadian Travelers Check	■ Canadian Travelers Check amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each Canadian Travelers Check entered. ■ <ARG> equals the number of the Canadian Travelers Check.
Canadian Gift Certificate Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Canadian Gift Certificate when Count Till at Reconcile equals Detail and the tender was received.
<ARG> Canadian Gift Certificate	■ Canadian Gift Certificate amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each Canadian Gift Certificate entered. ■ <ARG> equals the number of the Canadian Gift Certificate.
Canadian Store Credit Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Canadian Store Credit when Count Till at Reconcile equals Detail and the tender was received.
<ARG> Canadian Store Credit	■ Canadian Store Credit amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each Canadian Store Credit entered. ■ <ARG> equals the number of the Canadian Store Credit.
Mexican Gift Certificate Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Mexican Gift Certificate when Count Till at Reconcile equals Detail and the tender was received.
<ARG> Mexican Gift Certificate	■ Mexican Gift Certificate amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each Mexican Gift Certificate entered. ■ <ARG> equals the number of the Mexican Gift Certificate.
Mexican Store Credit Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Mexican Store Credit when Count Till at Reconcile equals Detail and the tender was received.

Table 15–25 (Cont.) Till Reconcile Event Components

Event Components	Notes
<ARG> Mexican Store Credit	■ Mexican Store Credit amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each Mexican Store Credit entered. ■ <ARG> equals the number of the Mexican Store Credit.
UK Gift Certificate Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered UK Gift Certificate when Count Till at Reconcile equals Detail and the tender was received.
<ARG> UK Gift Certificate	■ UK Gift Certificate amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each UK Gift Certificate entered. ■ <ARG> equals the number of the UK Gift Certificate.
UK Store Credit Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered UK Store Credit when Count Till at Reconcile equals Detail and the tender was received.
<ARG> UK Store Credit	■ UK Store Credit amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each UK Store Credit entered. ■ <ARG> equals the number of the UK Store Credit.
European Gift Certificate Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered European Gift Certificate when Count Till at Reconcile equals Detail and the tender was received.
<ARG> European Gift Certificate	■ European Gift Certificate amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each European Gift Certificate entered. ■ <ARG> equals the number of the European Gift Certificate.
European Store Credit Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered European Gift Certificate when Count Till at Reconcile equals Detail and the tender was received.
<ARG> European Store Credit	■ European Store Credit amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each European Store Credit entered. ■ <ARG> equals the number of the European Store Credit.
Japanese Gift Certificate Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Japanese Gift Certificate when Count Till at Reconcile equals Detail and the tender was received.

Table 15–25 (Cont.) Till Reconcile Event Components

Event Components	Notes
<ARG> Japanese Gift Certificate	■ Japanese Gift Certificate amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each Japanese Gift Certificate entered. ■ <ARG> equals the number of the Japanese Gift Certificate.
Japanese Store Credit Total	■ Entered tender amount if Count Till at Reconcile equals Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close equals No. ■ Total of all entered Japanese Store Credit when Count Till at Reconcile equals Detail and the tender was received.
<ARG> Japanese Store Credit	■ Japanese Store Credit amount entered when Count Till at Reconcile equals Detail. ■ There is an audit log entry for each Japanese Store Credit entered. ■ <ARG> equals the number of the Japanese Store Credit.
Blind Close	■ True. ■ False.
Transaction Number	Transaction number assigned to till reconcile.

Parameter Log Events

The following are parameter log events.

Modify Application Parameter

This is a Back Office, Point-of-Service and Central Office event.

This event is written to the audit log if the Settings For Audit Logging system setting equals INFO and the Modify Application Parameter system setting equals INFO.

- Event data collection starts when the operator selects a parameter to modify.
- Event data collection ends when the operator selects to save.

Table 15–26 Modify Application Parameter Event Components

Event Components	Notes
Event Name	Modify Application Parameter.
Event Status	■ Success. ■ Failure.
Event Originator	Class Name and Method Name (ClassName.methodName).
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at. Only applicable for Back Office.
User ID	User ID performing the event.
Parameter Group	Parameter Group.
Parameter Name	Name of the Parameter.

16

Oracle Retail Returns Management Customer Data Import

The Returns Management customer import feature is a way for a retailer to import a large amount of pre-existing customer data into the data-store accessed by Returns Management. Besides the usual customer information, such as Name, Address and Phone, this feature also enables the retailer to assign an *exception count* to a customer, based on third-party information about an individual (for example, information from credit bureaus, information about criminal records and so on). In Returns Management, higher exception counts are indicative of customers whose past behavior is of concern from a returns standpoint.

Most of the customer information imported is the same as the customer information sent in the Returns Management Return Request XML message. The XML schema definition of this information was contained in the `RM-ReturnRequest.xsd` file. In order that both schema files can share the same customer schema definitions, these definitions have been removed from `RM-ReturnRequest.xsd`, and placed in a new file:

`RM-CustomerInfo.xsd`

The Customer Import XML is defined by the following new schema file:

`RM-CustomerImport.xsd`

The xsd files can be found in the `<Install_DIR>/returnsmgmt/api/returnsSchemas.zip` archive.

The following is a listing of the `RM-CustomerImport.xsd` file:

Example 16–1 RM-CustomerImport.xsd

```
    <xsd:include schemaLocation="RM-CustomerInfo.xsd"/>
    <xsd:element name="ReturnsCustomers" type="ReturnsCustomersType"/>
    <xsd:complexType name="ReturnsCustomersType">
        <xsd:sequence>
            <xsd:element name="ReturnsCustomer" type="ReturnsCustomerType"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ReturnsCustomerType">
        <xsd:sequence>
            <xsd:element name="positiveID" type="PositiveID"/>
            <xsd:element name=”customerInfo" type="MoreCustomerInfo"/>
            <xsd:element name="exceptionCount" type="xsd:integer"/>
            <xsd:element name="notes" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
```

Notice that the `RM-CustomerImport.xsd` file refers to the `RM-CustomerInfo.xsd` file.

Example 16–2 RM-CustomerInfo.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:annotation>
        <xsd:documentation>
        Defines the PositiveID and MoreCustomerInfo elements.
        </xsd:documentation>
    </xsd:annotation>

    <xsd:complexType name="PositiveID">
        <xsd:sequence>
            <xsd:element name="number" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="type" minOccurs="1" maxOccurs="1">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="DriversLicense"/>
                        <xsd:enumeration value="MilitaryID"/>
                        <xsd:enumeration value="Passport"/>
                        <xsd:enumeration value="StateCard"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:element>
            <xsd:element name="issuer" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="issued" type="xsd:date" minOccurs="0"
maxOccurs="1"/>
            <xsd:element name="expiration" type="xsd:date" minOccurs="0"
maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="MoreCustomerInfo">
        <xsd:sequence>
            <xsd:element name="lastName" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="firstName" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="middleName" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
            <xsd:element name="gender" minOccurs="0" maxOccurs="1">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="Male"/>
                        <xsd:enumeration value="Female"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:element>
            <!-- format of yyyyMMdd -->
            <xsd:element name="birthDate" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
            <xsd:element name="address1" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="address2" type="xsd:string" minOccurs="0"
```

```
maxOccurs="1"/>
            <xsd:element name="city" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="state" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
            <!-- zip code-->
            <xsd:element name="postalCode" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="country" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="telephoneAreaCode" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
            <xsd:element name="telephoneLocalNumber" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>

</xsd:schema>
```

The following is an example of the Returns Management Customer Import XML file.

Example 16–3 RMCustomerImport.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ReturnsCustomers>

    <ReturnsCustomer>
        <positiveID>
            <EncryptedPositiveID>TX004</EncryptedPositiveID>
            <type>DriversLicense</type>
            <issuer>US_TX</issuer>
            <issued>2004-04-01</issued>
            <expiration>2012-04-01</expiration>
        </positiveID>
        <customerInfo>
            <lastName>TX004</lastName>
            <firstName>Joe</firstName>
            <address1>Some address line 1</address1>
            <address2>Some address line 2</address2>
            <city>Austin</city>
            <state>TX</state>
            <postalCode>78701</postalCode>
            <country>USA</country>
            <telephoneAreaCode>512</telephoneAreaCode>
            <telephoneLocalNumber>555-5100</telephoneLocalNumber>
        </customerInfo>
        <exceptionCount>100</exceptionCount>
        <notes>Test import</notes>
    </ReturnsCustomer>

    <ReturnsCustomer>
        <positiveID>
            <EncryptedPositiveID>TX002</EncryptedPositiveID>
            <type>DriversLicense</type>
            <issuer>US_TX</issuer>
        </positiveID>
        <customerInfo>
            <lastName>TX002</lastName>
            <firstName>Joe</firstName>
            <address1>Some address</address1>
```

```
            <address2>Some address2</address2>
            <city>Austin</city>
            <state>TX</state>
            <postalCode>78701</postalCode>
        </customerInfo>
        <exceptionCount>200</exceptionCount>
        <notes>Import TX002</notes>
    </ReturnsCustomer>

</ReturnsCustomers>
```

The following new parameter has been added for the customer import feature:

`ReturnsCustomerImportDuplicateRecordAction`

Customer data imported through this feature is stored in one or more of the tables identified in Table 16–1.

Table 16–1 Customer Information Tables

Table name	Information held in table
RM_CT	Returns Customer ID and Positive ID
RM_CT_ID	Returns Customer ID to Customer ID mapping
RM_CT_SCR	Customer exception count
RM_CT_SV_OVRD	Comments (notes) when exception count is changed
PA_CT	Customer ID
PA_PRTY	Customer ID to Party ID mapping
PA_CNCT	Customer last and first name
LO_ADS	Customer address
PA_PHN	Customer phone number

17

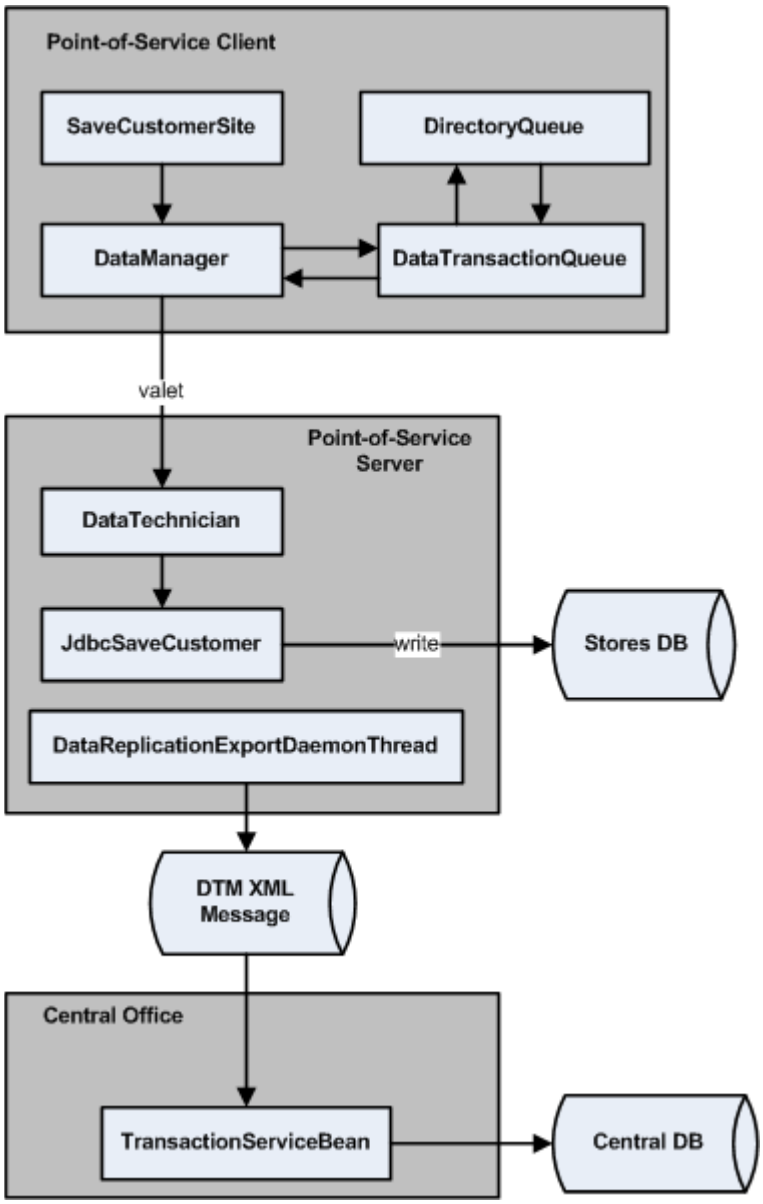
Centralized Customer

Centralized Customer enables a Central Office operator to enter and manage customer data. Centralized Customer also provides Point-of-Service the ability to retrieve customer information from a central database. This functionality enables Point-of-Service to support customer-specific pricing. Other Point-of-Service features support the need for Centralized Customer such as assisting in pickup and delivery orders, and obtaining Tax ID numbers for customers required to manage specific tax forms. Retail stores and cashiers also benefit from this functionality. Since customer information can be retrieved from a central database, customer information does not have to be re-entered at different stores.

The Centralized Customer functionality enables the operator to manage existing customer information and add new customers to the central database. The operator has the ability to search for a customer, modify existing customer information, or mark a customer's record for deletion from the database. The operator can also assign a Pricing Group to the customer which allows retailers the ability to offer customer specific pricing. Pricing Groups can be assigned to a Price Promotion or Discount Rule.

There are two types of customers: individual and business. Business customers require slightly different data than individual customers such as tax certificate numbers.

Figure 17–1 Centralized Customer Object Model



18

Automated E-Mail Messages

Fulfillment automatically creates e-mail messages for customers when certain conditions are met. Each transaction has a status associated with it. As each step in the order process is completed, the status is automatically updated to reflect these changes.

Whenever the order status changes to Filled, Partial, Completed, or Cancelled, an automatic e-mail message is created. The order information is inserted into an e-mail file and sent to the server.

The created e-mail messages are stored in the database. Point-of-Service does not send the e-mail messages to customers. The retailer is responsible for sending the e-mail messages.

The stored messages can be found in the table **DO_EMSG**.

The following table lists the status values for each e-mail message as found in the column **ST_EMSG**.

Table 18–1 E-Mail Message Status Values

Value	E-Mail Message Status
0	public static final int MESSAGE_STATUS_NEW
1	public static final int MESSAGE_STATUS_REPLIED
2	public static final int MESSAGE_STATUS_OUTBOX
3	public static final int MESSAGE_STATUS_SENT
4	public static final int MESSAGE_STATUS_READ

For more information about Automated E-Mail Messages and Service Alert, see the *Oracle Retail Point-of-Service User Guide*.

19

Register Cash Notification

Register Cash Notification gives retailers added security and enables stores to manage cash by register and till. Register Cash Notification informs Point-of-Service users when the amount of cash in the register or till is above or below a configurable amount as defined by a set of parameters. Register Cash Notification notifies the Point-of-Service user of the cash discrepancy through a modal message for cash warning over and a non-modal message for cash warning under.

For more information about Register Cash Notification, see the *Oracle Point-of-Service User Guide*.

Configuration

Edit the following configuration files as described to enable Register Cash Notification.

application.xml

The following configurations are maintained in the application parameters:

Table 19–1 Application Parameters

Configuration	Description
CashAmountOverWarningFloat	This configuration contains the upper limit of Cash amount, crossing which modal warning message will be displayed.
CashAmountUnderWarningFloat	This configuration will contain the lower limit of Cash amount; a non-modal warning message will be displayed for the same.

application.properties

Add the following timing parameters for Cash Warning UNDER to application.properties:

- CashDrawerWarning.AnimationDelay=7
- CashDrawerWarning.Lifetime=6000
- CashDrawerWarning.Waittime=500

dialogText_en.properties

Add the following parameters for the OVER Dialog Warning:

- DialogSpec.OverCashDrawerWarning.title=Cash Drawer Maintenance
- DialogSpec.OverCashDrawerWarning.description=Cash Drawer Warning
- DialogSpec.OverCashDrawerWarning.line3=Notify a manager
- DialogSpec.OverCashDrawerWarning.line8=Press Enter to continue

posText_en.properties

Add the following parameter for warning message when UNDER:

StatusPanelSpec.CashDrawerUnderWarningMessage= Contact the Manager

20

Receipt Builder

Receipt Builder is a tool that is used to maintain receipt formatting and content. Instead of the construction of receipt output existing in Java code, which requires Java programming knowledge to change, the construction of the receipt output is defined within easily edited XML files. The XML files can be displayed graphically and edited within the Receipt Builder editor or edited as plain text in any other editor. This externalizes the receipts in a way that is easily configurable and that does not require layers of code extensions. The XML contains a combination of static receipt elements that always print the same and other receipt elements that are dynamic. The dynamic elements print values obtained from Java objects that exist in memory at runtime. The XML also allows for a wide range of formatting of the obtained values in order to make them suitable for printing.

The Receipt Builder editor requires serialized (that is, persisted) sample data objects for the construction of new receipts. The data objects allow a developer to choose which Java methods are executed in order to obtain the data to print for a particular element. At Point-of-Service runtime, the object resides in memory and the new receipt printing framework executes the specified methods against the printing framework based upon the XML instructions. These serialized object files can be generated at Point-of-Service runtime.

Instructions on how to format the information into the fixed-width receipt printers is kept in XML blueprint files (also known as templates). These XML files are read at runtime and combined with the runtime in-memory objects to produce printable output. The Point-of-Service client caches the XML once it is read. Only if the client detects the blueprint file has a newer timestamp will the client reread the XML or if a new blueprint is sent from Central Office.

Note: For more informtaion, see the O*racle Retail Point-of-Service Receipt Builder Tool User Guide.*

This document is available through My Oracle Support. Access My Oracle Support at the following URL:

`https://support.oracle.com`

Oracle Retail Point-of-Service Receipt Builder Tool User Guide **(Doc ID: 1277454.1)**

Receipt Builder XML Blueprint Files

The XML blueprints are installed at <*source_directory*>\applications\pos\deploy\client\receipts.

The Point-of-Service client caches all blueprints once read. Upon further printing, if the blueprint file's timestamp has changed to a newer timestamp, the XML is reread. Additionally, if the client receives new blueprints using FileTransfer then the cache is also cleared.

Example XML Blueprint File

The following file is an example of an XML blueprint file that uses data from an object called "com.demo.Person". Various lines will be printed to the receipt such as "Name:" and "Birth Date:"

Example 20–1 Example.bpt

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint id="receipt.bpt" copies="1">
    <report name="Report" id="1">
        <group id="99401936">
            <line id="32962587">
                <imageElement fileName="ReceiptLogo.jpg"
idePath="/demo/bin/ReceiptLogo.jpg" id="25483246" justified="1" />
            </line>
            <line id="15081425">
                <dateTimeElement formatter="Date.SHORT" prefix="(" suffix=")"
id="4536570" />
                <element text=" " id="24559530" stretch="true" />
                <dateTimeElement formatter="Time.SHORT" id="6253254" />
            </line>
        </group>
        <group id="1">
            <line id="24595355">
                <element text="Idæ‰<:" id="25255986" />
                <methodElement id="10602994">
                    <method returns="String" name="getId" class="com.demo.Person"
/>
                </methodElement>
                <element text="Name:" id="1" justified="2" stretch="true" />
                <methodElement id="26482774">
                    <method returns="String" name="getName"
class="com.demo.Person" />
                </methodElement>
            </line>
            <line id="17089909">
                <element text="Height:" id="18455598" />
                <methodElement id="6311384">
                    <method returns="double" name="getHeight"
class="com.demo.Person" />
                </methodElement>
                <methodElement id="13946325">
                    <method returns="String" name="getLocalizedSalutation"
class="com.demo.Person" param="Locale" />
                </methodElement>
            </line>
            <line id="4171180">
                <element text="Age:" id="19840829" />
                <methodElement
```

```
formatter="0#New|1#Printed|2#Partial|3#Filled|4#Canceled|5#Completed|6#Voided"
id="32596007" escapeSequence="\|bC\|iC">
                    <method returns="int" name="getAge" class="com.demo.Person" />
                </methodElement>
                <methodElement formatter="#0;(#0)" id="16747213">
                    <method returns="BigDecimal" name="getBigNumber"
class="com.demo.Person" />
                </methodElement>
            </line>
            <line id="9818046">
                <element text="Sex:" id="14253732" fillChar="." stretch="true" />
                <methodElement valuePrintedWhenFalse="female"
valuePrintedWhenTrue="male" id="6446153" stretch="true">
                    <method returns="boolean" name="isSex" class="com.demo.Person"
/>
                </methodElement>
            </line>
            <line id="24763620">
                <element text="Salary:" id="15358832" />
                <methodElement id="25586725">
                    <method returns="CurrencyDecimal" name="getSalary"
class="com.demo.Person" />
                </methodElement>
            </line>
            <line id="26542488">
                <element text="Nick name:" id="19086511" />
                <methodElement prefix="&amp;quot;" suffix="&amp;quot;" id="27541747">
                    <method returns="String" name="getNickname"
class="com.demo.Person" />
                </methodElement>
            </line>
            <line id="26980954">
                <element text="Birth Date:" id="26154958" />
                <methodElement formatter="Date.MEDIUM" id="12290792">
                    <method returns="Date" name="getBirthDate"
class="com.demo.Person" />
                </methodElement>
            </line>
            <line id="14314484">
                <element text="Spouse:" id="16920240" />
                <methodElement fixedWidth="20" id="15369072" fillChar="%"
justified="2">
                    <method returns="Person" name="getSpouse"
class="com.demo.Person">
                        <method returns="String" name="getName"
class="com.demo.Person" />
                    </method>
                </methodElement>
            </line>
            <line id="31820984">
                <element text="" id="5367480" fillChar="_" stretch="true" />
            </line>
            <line id="24744797" />
            <line id="12182618">
                <element text="Relatives" id="4387753" />
            </line>
            <line id="4126736">
                <element text="" id="19625657" fillChar="-" stretch="true" />
            </line>
        </group>
```

```
        <group id="18541827">
            <line id="21925102" dependsOnPresenceOf="28217713">
                <element text="    Name:" id="8930268" />
                <methodElement id="28217713">
                    <method returns="List&lt;Person&gt;" name="getRelatives"
class="com.demo.Person">
                        <method returns="java.lang.Object[]" name="toArray"
class="java.util.List">
                            <method returns="String" name="getName"
class="com.demo.Person" />
                        </method>
                    </method>
                </methodElement>
            </line>
            <line id="421988">
                <element text="     Age:" id="20121217"
dependsOnPresenceOf="28007313" />
                <methodElement fixedWidth="4" formatter="##0.##E0"
printedWhenValueZero="false" id="28007313">
                    <method returns="List&lt;Person&gt;" name="getRelatives"
class="com.demo.Person">
                        <method returns="java.lang.Object[]" name="toArray"
class="java.util.List">
                            <method returns="int" name="getAge"
class="com.demo.Person" />
                        </method>
                    </method>
                </methodElement>
            </line>
        </group>
        <group id="3818530">
            <line id="9949215">
                <element text=" " id="4513709" />
            </line>
            <line id="14721926">
                <element text="Should not print on the third copy" id="19625657"
dependsOnPresenceOf="23450220" />
            </line>
            <line id="1043272">
                <methodElement fixedWidth="42" id="19570995"
printedAsBarcode="true" justified="1">
                    <method returns="int" name="hashCode" class="java.lang.Object"
/>
                </methodElement>
            </line>
        </group>
        <dependsOn returns="boolean" name="isSex" class="com.demo.Person" />
    </report>
    <linkReport documentType="footer" idePath="/demo/receipts/footer.bpt"
id="12856042" />
</blueprint>
```

Receipt Builder XSD

The following XSD defines the structure of the XML blueprints:

Example 20–2 Receipt Builder XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.org/receipt"
xmlns:tns="http://www.example.org/receipt"
elementFormDefault="qualified">

    <element name="blueprint" type="tns:blueprintType" />

    <complexType name="blueprintType">
    <annotation><documentation>
   This element represents the a single receipt blueprint. A blueprint
            is the instructions for single transactional interaction with the
            Point-of-Service printer. A blueprint can consist of one to many
reports.
    </documentation></annotation>
    <choice>
    <element name="report" type="tns:reportType" minOccurs="0"
maxOccurs="unbounded" />
            <element name="linkReport" type="tns:linkReportType" minOccurs="0"
maxOccurs="unbounded" />
    </choice>
    <attribute name="id" type="string" use="required" />
    </complexType>

    <complexType name="abstractReportType">
        <annotation><documentation>
            This is a super type for generic reports or linkReports. There can
            be one or many reports for a single receipt blueprint. The receipt
            paper is cut after the report depending on the attribute cutPaper.
            A report can be repeated for multiple copies depending on the
            attribute copies. It is possible for an entire report to not be
            printed based upon if the results of the dependsOn element are null
            or false.
        </documentation></annotation>
        <sequence>
            <element name="dependsOn" type="tns:methodType" minOccurs="1"
maxOccurs="1"/>
        </sequence>
        <attribute name="id" type="string" use="required" />
        <attribute name="copies" type="int" default="1" />
        <attribute name="cutPaper" type="boolean" default="true" />
    </complexType>

    <complexType name="reportType">
        <annotation><documentation>
            A report is a collection of receipt groups where usually the receipt
            paper is cut after the report. The name of the report is currently
            only for informational purposes.
        </documentation></annotation>
    <complexContent>
    <extension base="tns:abstractReportType">
        <sequence>
            <element name="group" type="tns:groupType" minOccurs="1"
maxOccurs="unbounded" />
        </sequence>
```

```
<attribute name="name" type="string" />
</extension>
</complexContent>
</complexType>

<complexType name="linkReportType">
    <annotation><documentation>
        A "link report" is meant as a signal to print an external blueprint
        that can be identified by the documentType. It does not contain
        groups.
    </documentation></annotation>
    <complexContent>
        <extension base="tns:abstractReportType">
            <attribute name="documentType" type="string" use="required" />
        </extension>
    </complexContent>
</complexType>

<complexType name="groupType">
<annotation><documentation>
        A group is a logical collection of receipt lines that can be
        repeated if they contain an iteration or array. The group contains
        lines that are printed in order. If one of the lines contains a
        method stack from a methodElement and that method stack has a method
        call which executes on an array or collection then the entire group
        of lines will be repeated for the length or size of the iteration.

        NOTE: including method stacks which contain different arrays does
        not make sense and will not work as expected. MethodElements within
        the group should reference the same array or collection.

        Sub-iterations, i.e. arrays within arrays can be handled, but only
        the line that contains the sub-array will be repeated for the length
        of that array. The the entire group will repeat for the outer array.
</documentation></annotation>
<choice>
<element name="line" type="tns:lineType" />
</choice>
<attribute name="id" type="string" use="required" />
</complexType>

<complexType name="lineType">
    <annotation><documentation>
        A line is a collection of elements that ends with a carriage return.
        Lines will be repeated (even within a repeating group) if they
        contain a methodElement which executes on an array within an array.

        It is possible for an entire line including the carriage return to
        not be printed based upon if the results of the dependsOnPresenceof
        methodElement are null or false.  The value is the id of a
        methodElement in this group.
    </documentation></annotation>
    <choice>
        <element name="element" type="tns:elementType" />
        <element name="copyElement" type="tns:copyElementType" />
        <element name="imageElement" type="tns:imageElementType" />
        <element name="methodElement" type="tns:methodElementType" />
        <element name="dateTimeElement" type="tns:dateTimeElementType" />
    </choice>
    <attribute name="id" type="string" use="required" />
```

```
        <attribute name="dependsOnPresenceof" type="string" />
    </complexType>

<!-- Parent type for all elements. -->
    <complexType name="abstractElementType">
        <annotation><documentation>
            Provides a super type for attributes that are the same between all
            elements. An element's output can be affected by the JPOS
            escapeSequence attribute. An element can be printed as a bar code
            for scanning depending on the printedAsBarCode attribute. The
            attribute justify controls whether the text in the element is
            justified left "0", center "1", or right "2" in its space.

            It is possible for an element to not be printed based upon if the
            results of the dependsOnPresenceof attribute are true or false. The
            value is the id of a methodElement in this group.
        </documentation></annotation>
    <attribute name="id" type="string" use="required" />
    <attribute name="dependsOnPresenceof" type="string" />
    <attribute name="escapeSequence" type="string" />
    <attribute name="printedAsBarCode" type="boolean" default="false" />
 <attribute name="justified" type="int" />
    </complexType>

<!-- Type for static text elements. -->
    <complexType name="elementType">
        <annotation><documentation>
            Represents an element in the receipt report that can display static
            text and its width can be stretched to take up extra space.
        </documentation></annotation>
    <complexContent>
    <extension base="tns:abstractElementType">
    <attribute name="text" type="string" use="required" />
    <attribute name="stretch" type="boolean" default="false" />
    </extension>
    </complexContent>
    </complexType>

<!-- Type for text elements that change for each copy of a report. -->
    <complexType name="copyElementType">
        <annotation><documentation>
            Represents an element in the receipt report that can display text
            that changes based upon the index of the current report being
            printed and its width can be stretched to take up extra space.
        </documentation></annotation>
    <complexContent>
    <extension base="tns:abstractElementType">
    <sequence>
    <element name="copyText" type="string" minOccurs="1" maxOccurs="unbounded" />
    </sequence>
    <attribute name="stretch" type="boolean" default="false" />
    </extension>
    </complexContent>
    </complexType>

<!-- A static element that displays an image. -->
    <complexType name="imageElementType">
        <annotation><documentation>
            Represents an element in the receipt report that can display an
            image. The image file is expected to be in the present working
```

```
            directory of the application, e.g. bin/. The image will be centered
            in its line and will be the only element in its line. The idePath
            specified is only useful to the Receipt Builder plug-in for finding
            the file in Eclipse.
        </documentation></annotation>
    <complexContent>
    <extension base="tns:abstractElementType">
    <attribute name="fileName" type="string" use="required" />
    <attribute name="idePath" type="string"/>
    </extension>
    </complexContent>
    </complexType>

<!-- Parent type for object and method elements. -->
    <complexType name="javaElementType">
        <annotation><documentation>
            Represents an element in the receipt report that displays text based
            upon the value of a Java object. The output can be formatted with a
            space, prefix and suffix, given a fixed width padded by spaces or
            given a format pattern string that conforms to Java types:
            java.text.SimpleDateFormat, java.text.DecimalFormat,
            java.swing.text.MaskFormatter, or java.text.ChoiceFormat.
        </documentation></annotation>
    <complexContent>
    <extension base="tns:abstractElementType">
    <attribute name="precededBySpace" type="boolean" default="true" />
 <attribute name="fixedWidth" type="int" />
    <attribute name="formatter" type="string" />
    <attribute name="prefix" type="string" />
    <attribute name="suffix" type="string" />
    </extension>
    </complexContent>
    </complexType>

<!-- An element that specifies the printing of the current date/time. -->
    <complexType name="dateTimeElementType">
        <annotation><documentation>
            Represents an element in the receipt report that displays the
            current date or time. The output can be formatted by
            java.text.SimpleDateFormat.
        </documentation></annotation>
    <complexContent>
    <extension base="tns:javaElementType"/>
    </complexContent>
    </complexType>

<!-- An element that contains the execution of of a stack of methods. -->
    <complexType name="methodElementType">
        <annotation><documentation>
            Represents an element in the receipt report that displays the
            value of a method call. If the method returns a boolean, then
            specific values can be printed instead of "true" or "false". If the
            element should not be printed when the output is null or empty
            string, then printedWhenLengthZero should be false. If the output is
            a number that should not be printed when it is zero, then
            printedWhenValueZero should be false.
        </documentation></annotation>
    <complexContent>
    <extension base="tns:javaElementType">
    <sequence>
```

```
    <element name="method" type="tns:methodType" minOccurs="1" maxOccurs="1"/>
    </sequence>
    <attribute name="valuePrintedWhenFalse" type="string" default="false" />
    <attribute name="valuePrintedWhenTrue" type="string" default="true" />
    <attribute name="printedWhenValueZero" type="boolean" default="true" />
    <attribute name="printedWhenLengthZero" type="boolean" default="false" />
    </extension>
    </complexContent>
    </complexType>

<!-- Information of a method call. -->
    <complexType name="methodType">
        <annotation><documentation>
            This is a method in a method call stack. It can have a child method.
        </documentation></annotation>
        <sequence>
            <element name="method" type="tns:methodType" minOccurs="1"
maxOccurs="1"/>
        </sequence>
    <attribute name="returns" type="string" use="required" />
    <attribute name="name" type="string" use="required" />
        <attribute name="param" type="string" use="optional" />
    <attribute name="class" type="string" use="required" />
    </complexType>

</schema>
```

Configuration

The following information aids in configuring the receipt builder application.

Conduit Configuration

In <*source_directory*>\applications\pos\deploy\client\config\conduit\ClientConduit.xml, the manager for PrintableDocumentManager should be set to **BlueprintedDocumentManager**.

```
<MANAGER name="PrintableDocumentManager"
         package="oracle.retail.stores.pos.receipt.blueprint"
         class="BlueprintedDocumentManager">
<PROPERTY propname="configScript"
propvalue="classpath://config/manager/BlueprintedDocumentManager.xml" />
</MANAGER>
```

> **Note:** Ensure <*source_directory*>\applications\pos\deploy\client\config\manager\BlueprintedDocumentManager.xml is present and configured.

Manager Configuration

The BlueprintedDocumentManager takes a configuration file location as a property value. The default value is specified above. This configuration file can specify the flag of whether the beans will be persisted at print time. The other property is the directory the receipt blueprints can be found in. This is the same directory in which the persisted beans are placed.

In this configuration file, the base blueprint file name is mapped to the document type, enabling a different blueprint file to be printed for the specified document type. For example, if a different sale receipt is to be printed other than the released version, a different file name such as **MySaleReceipt.bpt** can be specified. Note that the locale-specific naming convention still takes place. This means that on an American English Point-of-Service client, the file SaleReceipt_en.bpt will be searched first, followed by SaleReceipt_en.bpt, then SaleReceipt.bpt.

```
<RECEIPT type="OrderReceipt" fileName="OrderReceipt.bpt" />
<RECEIPT type="RedeemReceipt" fileName="RedeemReceipt.bpt" />
<RECEIPT type="SaleReceipt" fileName="SaleReceipt.bpt" />
<RECEIPT type="SendGiftReceipt" fileName="SendGiftReceipt.bpt" />
```

Spring Configuration

The implementation of BlueprintedReceipt is set by Spring. Changing this class can change how the blueprint instructions are used to create printer output.

This setting can be found in the ApplicationContext.xml file found in the pos/config/context directory.

```
    <!-- Blueprint receipt printing. Class must extend
oracle.retail.stores.pos.receipt.blueprint.BlueprintedReceipt -->
    <bean id="application_BlueprintedReceipt" class=
"oracle.retail.stores.pos.receipt.blueprint.BlueprintedReceipt"
          lazy-init="true"/>
```

Receipt Messages

The following information describes updating messages on receipts.

Updating the Legal Statement of Liability on a Receipt

There are two ways to update the legal statement of liability. One is to simply change the text "Legal statement of liability" to that which is desired. However, if the text is changed to "New legal statement of liability" the results would be as follows:

For a Sale:
Sale New legal statement of liability

For a Return:
Return New legal statement of liability

For a Layaway:
Layaway New legal statement of liability

For an Exchange:
Exchange New legal statement of liability

Note how **Sale**, **Return**, **Layaway** and **Exchange** seem to be hard-coded. Plus, they all share the same text. Below is a snippet of how CreditSignatureSlipReceipt_en.bpt would look for the above:

```
            <line id="12894866">
                <methodElement formatter="1#Sale|2#Return|5#Exchange|18#House
Account Payment|19#Layaway" id="31782456">
                    <method returns="int" name="getTransactionType"
class="oracle.retail.stores.pos.receipt.ReceiptParameterBeanIfc" />
                </methodElement>
                <element text=" New legal statement of liability" id="29789630" />
            </line>
```

The second way to change the text is to add the new text in place of Sale, Return, Layaway, Exchange, and so forth, as shown in the following example:

```
            <line id="12894866">
                <methodElement formatter="1#Sale Sig Slip Legal Statement|2#Return
Sig Slip Legal Statement|5#Exchange Sig Slip Legal Statement|18#House Account Sig
Slip Legal Statement|19#Layaway Sig Slip Legal Statement" id="31782456">
                    <method returns="int" name="getTransactionType"
class="oracle.retail.stores.pos.receipt.ReceiptParameterBeanIfc" />
                </methodElement>
                <element text="" id="29789630" />
            </line>
```

Note that the original element text in this example has been replaced by just "" (this element can be removed entirely). The advantage to this method is that the user can specify custom text for each transaction type, as well as eliminate the words **Sale**, **Return**, **Exchange**, and so forth, in the statement if they so desire.

Item Level Receipt Messages

Item Level Receipt Messages (ILRM) informs the cashier or the customer about the item in the Sell Item screen. This utility facilitates Item level messages on the screen, providing information to the cashier and the customer about the product or about certain attributes associated with the item. It also provides a facility to print the Item messages on the receipt either below the item or at the footer. Different messages can be configured for different types of receipts, or depending on whether the item is being sold or returned.

Rebate Receipt

A rebate receipt will be printed only if a rebate message exists for a given item. The blueprint is called RebateReceipt.bpt. If the item message exists, the XML Receipt Framework will invoke the getItemMessage() method from PLUItem object and display the rebate message. The item related information will invoke the same methods as invoked by the item information Group in SaleReceipt.bpt.

21

Scan Sheet

The scan sheet can be used to provide cashiers with a list of barcodes for items that are too small to have a label or sticker with a barcode, or for a service that carries a charge to the customer but is not tangible and therefore does not contain a sticker or label with a barcode. This functionality is an on-screen scan sheet. The scan sheet is accessed from the Sell item screen.

The scan sheet is represented as a grid. The retailer can configure each square of the grid to their specific needs. For example, one square might contain an icon representing alterations. Once selected, the next step is to go to the alternations detail screen so the user can capture the details needed to complete the alteration for the customer. If the user selects a square that contains an icon for an item, the user is returned to the Sell Item screen, and the item is added to the transaction.

Configuration

To use a scan sheet, you must set up the following properties.

Application.properties

Edit the following in the `<source_directory>\applications\pos\deploy\shared\config\application.properties` file:

- enableScanSheet – The valid values for this property are **true** and **false**. If the value is set to **true**, the scan sheet button is displayed. If the value is **false**, the scan sheet button is not displayed on the Sell Item screen.
- maxGridSize – The valid value for this property is any number. If it is set to **2**, then a grid of 2 by 2 is displayed. If set to **4**, a grid of 4 by 4 is displayed.

> **Note:** The suggested maximum grid size is 4 by 4. Any parameter value greater than 4 can lead to the display being distorted.

22

Serial Numbers

Serial numbering is a system used by manufacturers to be able to trace the history of any finished good that reaches the customers. When customers complain of defective goods, knowing the serial number enables the manufacturer to find out where the raw materials were purchased, who was involved in each production step, as well as which distributors the goods were channeled by.

Retailers that sell such high-valued or high-risk items have to track unique numbers or attributes for a single item or a group of items. This enables the retailer to have a tight control over every unit of every item in the inventory. The sale/return process needs to capture the serial number of the items, reserve/reverse status of item in Store Inventory Management and transmit the serial number to mark the item as sold to Store Inventory Management. The serial number of the sold item will also need to be transmitted with the transaction data to all the downstream applications that require Point-of-Service transaction data.

Point-of-Service will need to support sale of serial controlled items. The overall processing of a serial controlled item is broken into the following two parts:

- Serial Number Validation: When an item is scanned, if the UIN-required flag is set to **Yes**, the user is prompted for the serial number. If the UIN capture time is set to **StoreReceiving**, then the serial number is validated from Store Inventory Management.
- Serial Number Status Update: Serial number status is updated in Store Inventory Management based on the stock movement. All the transactions listed in the validation step are sent to Store Inventory Management for update.

Notes:

- The item scan process in the transaction listed for the serial number validation process prompts the user for the serial number if the item is a serial controlled item. The serial number is validated and upon the completion of the transaction, the inventory is reserved in Store Inventory Management.
- On completing the transactions listed in serial number update process, the serial number is transmitted as part of the transaction information to all downstream applications such as to Central Office, a sales audit application, Store Inventory Management, and so on.
- Point-of-Service handles the scenarios when Store Inventory Management is offline.

Configuration

The following functionality can be configured to enable the serial numbers functionality.

Enabling or Disabling Serialization Functionality

The property **SerializationEnabled=false** in `application.properties` file controls enabling or disabling of the feature. The Point-of-Service client installer sets the value **true** or **false** based on whether the user selects the serialization functionality.

Enabling or Disabling IMEI Functionality

The property **IMEIEnabled=false** in `application.properties` file controls enabling or disabling of the feature. This feature is not set by the installer and needs to be configured post-installation.

23

Oracle Retail Returns Management Exception File

The Oracle Retail Returns Management exception file is created and maintained by Returns Management for use in detecting and preventing fraud at the point-of-return. The exception file acts as a constantly evolving knowledge base that can help the Authorization Engine decide which customers, items, cashiers, or stores are at higher risk for return fraud.

The exception file holds an exception counter for a customer that is incremented based on suspicious return activity. If an activity is selected for inclusion in the exception counter, the system adds **1** to the exception count for each suspicious shopping activity. Likewise, activities can be configured for cashiers.

Exceptions and counting are based on real-time refund attempt activities occurring at the point-of-sale or return using the return result message that is sent by the point-of-sale or return to Returns Management at the conclusion of a transaction with an attempted refund. Return activities include activities that increment counters such as a return transaction by the customer without a receipt and with no retrieval of the original transaction, five same day returns as purchases within the last three days, and three returns today. In turn, activity thresholds might be breached and counting generated based on those thresholds.

The exception file holds an entry for each factor that triggers a count addition.

Exception File and Count Calculation

This section describes exception file count calculation for the following:

Customer

- Customer positive ID consisting of ID type, number, and issuer
- Exception count

Cashier

- Cashier ID
- Exception count

Exceptions Triggered

- Exception (the return activity that was breached)
- Target of the return activity (the customer ID, cashier ID)
- Date/Time of the exception

Note: Returns Management requires the use of a unique cashier ID for exception tracking.

Definition of Return, for Calculation

A return, for purposes of calculation, refers to an attempted return of a line item quantity.

Note: Returns count by type at the transaction level (unique line item level). This accommodates variations between points of sale that allow mixed situations or inherently disallow mixed situations. Counting at the quantity level could abnormally inflate exceptions, for example, returning a quantity of 8 china plates. Counting at the transaction level could exclude appropriate return or non-return counts due to the ability to mix returns from multiple original receipted transactions, or no receipt, within one Returns Management point-of-sale transaction.

Counting unique exceptions at the transaction level is conducted so that the customer is not penalized twice for the same situation within one transaction. If an exception occurs multiple times in a single transaction, that is counted as a single exception. For example, if a customer returns three different items without a receipt in a single return transaction, only one exception is generated.

Table 23–1 offers exception counting examples.

Table 23–1 **Exception Counting Examples**

Scenario	Exceptions Counted
Return attempt for 5 different items: ■ 1111 quantity 1 without receipt (no original transaction retrieved) ■ 2222 quantity 1 without receipt ■ 3333 quantity 1 without receipt ■ 4444 quantity 1 without receipt ■ 5555 quantity 1 with receipt (original transaction retrieved)	1 count for without a receipt 1 count for with a receipt
Return attempt for 4 different items, total quantity 5: ■ 1111 quantity 1 without receipt ■ 2222 quantity 2 without receipt ■ 3333 quantity 1 without receipt ■ 5555 quantity 1 with receipt	1 count for without a receipt 1 count for with a receipt

Table 23–1 (Cont.) **Exception Counting Examples**

Scenario	Exceptions Counted
Return attempt for 2 different items: ■ 1234 attempted return quantity of total 4 with two different receipts. Quantity 2 comes from original transaction 042419999999 and another quantity 2 comes from 042418888888. Split as two separate lines in Returns Management's point-of-sale because they would be selected from two different original transactions. ■ 5555 quantity 1 without a receipt	1 count for without a receipt 1 count for with a receipt
Two renter line items and one non-renter line item on the same receipted return attempt: ■ One item from the renter file is returned in the renter time frame, resulting in potential authorization, override, or denial. ■ A second item from the renter file is returned in the renter tomfooleries, resulting in potential authorization, override, or denial. ■ Another item, not listed in the renter file, is being returned and is sent to Returns Management for evaluation, resulting in authorization, override, or denial. The receipt is older than a parameterized number of days old (hence an expired receipt).	1 renter return 1 expired receipted return
Can mix returns both with and without receipts in the same transaction, as well as returns received from multiple transactions. In one return transaction, six line items could consist of: ■ Two returns without receipt—sample: resolve this one to authorized ■ Two receipted—quantity available—sample: resolve this one to authorized	1 count for without a receipt 1 count for with a receipt

Exceptions

The following are types of returns exceptions.

Customer Exceptions

Customer Exceptions can be flagged as behaviors that are tracked in the application, for use in Return Policies, using the Customer Exceptions to Track screen.

An exception is any activity that can be discerned from Return Ticket data, such as a non-receipted return, a return of an item contained in the Return Pattern Watch file, or a particular type of refund transaction such as a Price Adjustment.

When a customer exception occurs, a record is written to the exception file and the activity is available for research on that customer using the Customer Exception Search and Customer Exception Search Results screens.

The total number of exceptions that have occurred can be checked using a rule that can be included in return policies.

All of the exceptions are based on return ticket data.

Cashier Exceptions

Cashier Exceptions can be flagged as behaviors that are tracked in the application, for use in Return Policies, using the Cashier Exceptions to Track screen.

When a cashier exception occurs, a record is written to the exception file and the activity is available for research on that cashier using the Cashier Exception Search and Cashier Exception Search Results screens.

The total number of exceptions that have occurred can be checked using a rule that can be included in return policies.

All of the exceptions are based on return ticket data.

Cashier in this case is considered to be anyone captured as the employee on the return ticket, regardless of whether they have a cashier, associate manager, manager, or other store role.

Glossary

Batch

A collection of data operations that are processed during import at one time. The size is determined by a configurable parameter. Upon failure, an entire batch of data operations is rolled back.

Bundle

A collection of import files, one file per data type, stored as a compressed archive containing a manifest. It is expected that the retailer or implementation team is responsible for delivering to the store the bundle along with manifest for all data feeds to the store. MOM applications can package the bundle but do not provide delivery functions.

Corporate

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

Data Access Object (DAO)

A Java class that can retrieve and persist data to and from a data source. DAO is well-known JEE development pattern.

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. Specifically, the background data import mechanism supported by Back Office, Central Office and Returns Management.

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

JEE/J2EE

Java 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).

Limit (discount rule)

The maximum price allowed for a source or target to be part of a deal. Used most often when the source or target is a classification or department where many different priced items exist.

Manifest

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

Minimum Data

Minimum data is defined as the minimum set of data necessary to support the deployment of Stores applications.

If the user attempts to select any function or log in, an error may occur in the application without sample data loaded. See Sample Data.

Operational Data Store (ODS)

The corporate data repository that services Central Office.

POS Suite

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

Sample Data

A set of data used to demonstrate application features.

Store Applications

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

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

Even though Central Office and Returns Management run in the corporate environment, they are classified as store applications.

Store Database (SDB)

The data repository for store applications.

Threshold (discount rule)

The minimum price allowed for a source or target to be part of a deal. Used most often when the source or target is a classification or department where many different priced items exist.

Index

A

B

C

D

E

G

H

I

L

M

O

P

R

S

T

U