Retail passion. Innovative solutions.

Retek® Point-of-Sale™ 11.0

Operations Guide

Corporate Headquarters:

Retek Inc.
Retek on the Mall
950 Nicollet Mall
Minneapolis, MN 55403
USA

888.61.RETEK (toll free US)
Switchboard:
+1 612 587 5000

Fax:
+1 612 587 5100

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

Sales Enquiries:
+44 (0)20 7563 46 46

Fax:
+44 (0)20 7563 46 10

Printed in the United States of America.

Customer Support

Customer Support hours

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

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

Contact Method	Contact Information
E-mail	support@retek.com
Internet (ROCS)	rocs.retek.com Retek's secure client Web site to update and view issues
Phone	+1 612 587 5800

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

Australia	+1 800 555 923 (AU-Telstra) or +1 800 000 562 (AU-Optus)
France	0800 90 91 66
Hong Kong	800 96 4262
Korea	00 308 13 1342
United Kingdom	0800 917 2863
United States	+1 800 61 RETEK or 800 617 3835
Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

When contacting Customer Support, please provide:

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

RPOS release methodology

Retek's overall objective has always been to provide the customer with source code to any RPOS components that can be affected by a change in a customer's business requirements. Retek provides a clean separation between RPOS application core source code and business source code. This separation allows the deployment of future maintenance releases of core source code without affecting customers existing modifications.

The standard Retek license agreement includes access to source code for extension and modification to the following RPOS business type components including, but not limited to: credit authorization, collections, customer, employee, end-of-day process, online forms, goaling, item, layaway, paidouts, payments, transactions, pricing, readings, receipt, register, rules, scheduling, start-of-day process, store, GUI screen management (including internationalization language conversions), tax, timecard, transfer, network implementations, database implementations. In addition, the RPOS framework is extendable and a customer may create new business components to add to the RPOS product.

The standard Retek license agreement does not include source code for RPOS Application Tools, Retek developed peripheral device driver code or Middleware Application Core Server code. The Middleware Application Core Server code includes, but is not limited to the following business type core components: International currency calculations methods, GUI Screen Manager, Configuration Manager, Downtime Transaction Monitor, Peer-to-Peer Management, Multimedia Manager, Error Log Manager, Manifest Deployment Tool, EJB Manager. However, not having access to source code does not prevent a customer from extending or modifying these core business components. A customer may extend these components based on the Javadoc documentation generated from these Java class files, but any such change would not be supported under the standard maintenance agreement for the RPOS product.

Retek is willing to negotiate standard license agreements, maintenance agreements or escrow fees associated with access to middleware application server core code to alleviate any concerns a customer may have.

Contents

Chapter 1 – Introduction

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

Overview–what is RPOS?

The RPOS solution delivers one of the most powerful, scalable, flexible and stable point-of-sale solutions available. RPOS offers the following key components of functionality and more:

- Point-of-sale
- Cash management
- Customer management
- Labor management

Building on retail experience and research, the technology and retail business logic of RPOS adapts to a retailer’s business style, operating with existing networks, data repositories, hardware and operating systems.

Functional and technical capabilities

The system offers the following functional and technical capabilities:

- The Java-based N-tier architecture is designed for high availability, with built-in redundancies. The multi-tiered platform adapts to changes in the environment from hardware additions to database changes, enhancing system life. Retailers can adapt the architecture to their business needs and can change the number of tiers without changing code. The system's fit-client and network computer design make system upgrades, and price and inventory changes quick and easy, resulting in a lower cost of ownership. The separation of presentation, business logic, and data makes the software cleaner, more maintainable, and easier to modify.
- RPOS allows a retailer to extend and modify point-of-sale (RPOS) functionality. With integrated application tools such as RPOS Tools Application Builder, RPOS Tools Mission Control, and RPOS Tools Receipt/Report Builder, RPOS expands the definition of a POS system by allowing you to customize the application to meet all of your business needs. The Mission Control management tool allows for the maintenance of multiple application servers and back-end services, locally or remotely.
- Run-time recovery is possible from a fault on any tier. The system includes an online and an offline mode. Those transactions that are made during the offline mode are queued on the local terminal and automatically posted in chronological order, when the system's comes back online.
- A business object and rule framework facilitates data validation.
- An intuitive graphical-user interface (GUI) provides a comprehensive list of retail functionalities, along with online help and tips.
- Integration to backend applications and data repositories can be customized. Data is delivered through a common gateway. The system's messaging framework allows clients (cash registers, for example) to receive messages from and/or to send messages to topics and queues on a JMS server. RPOS supports multi-channel integration and handles large workloads while maintaining peak performance.

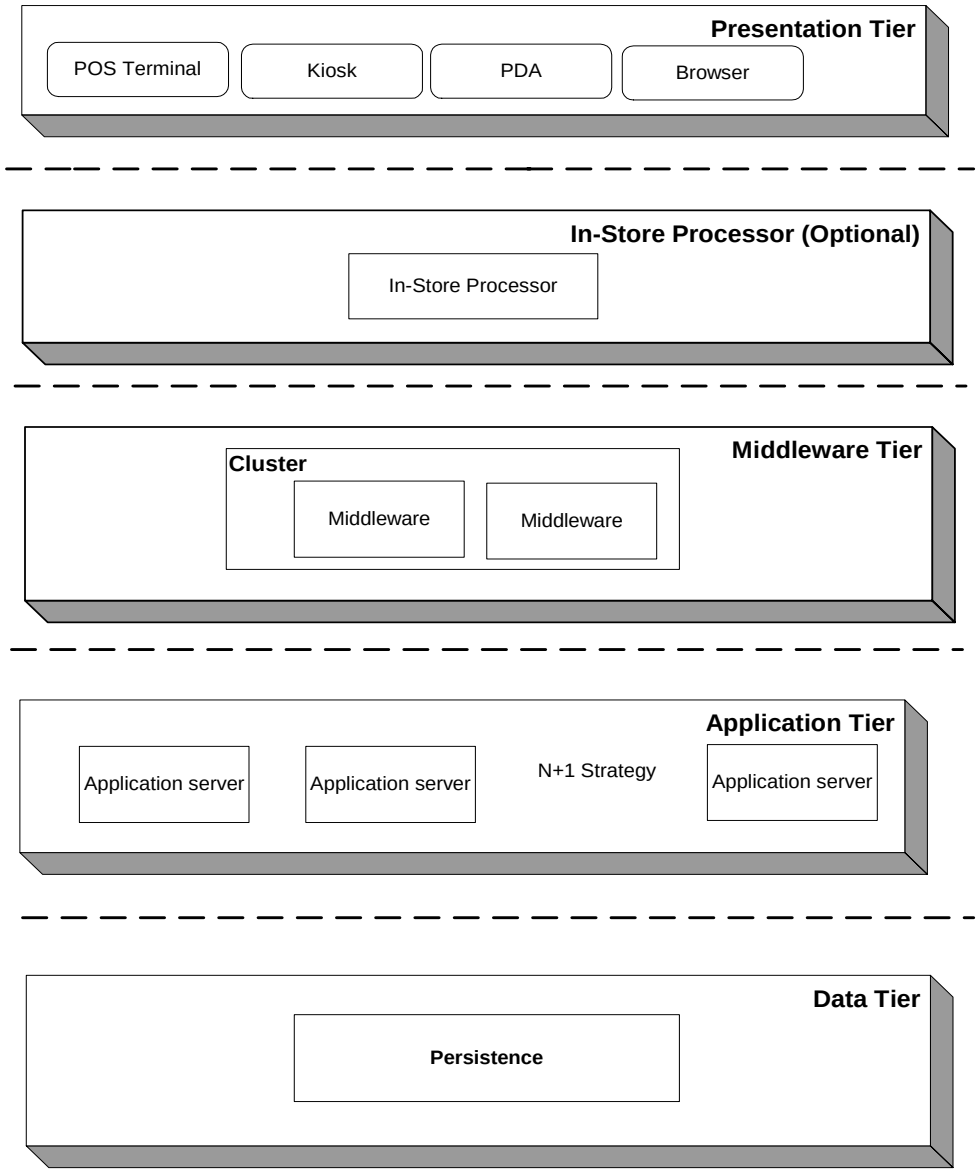
Who this guide is written for

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

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

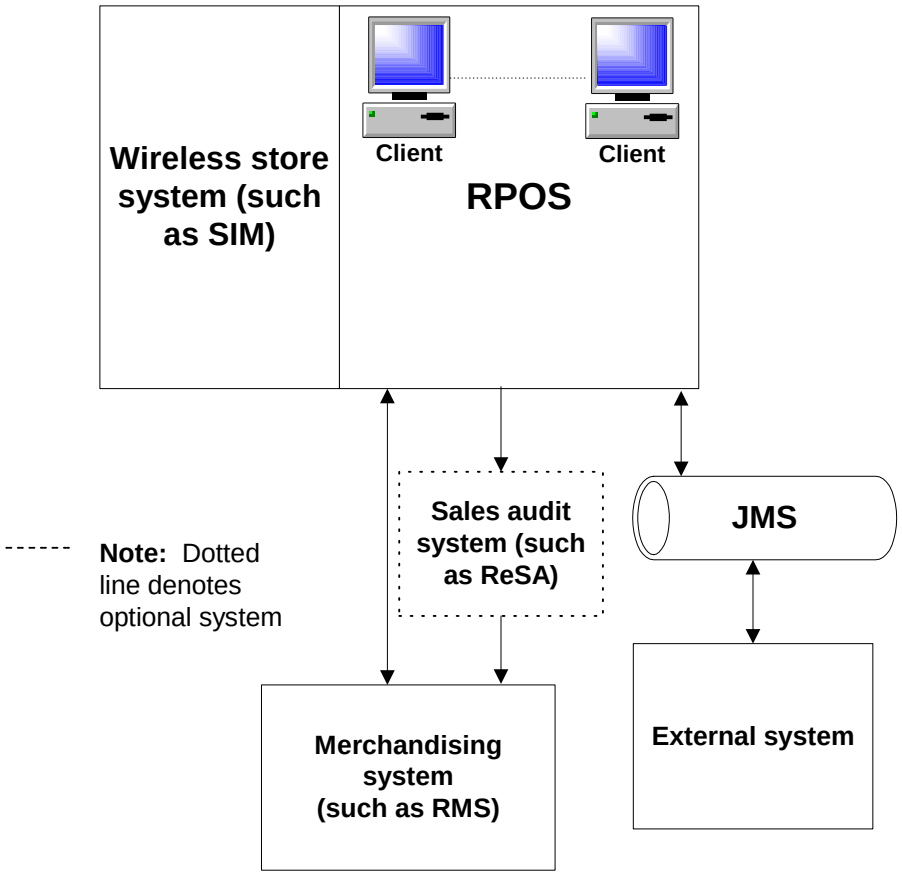
Technical architecture overview

RPOS's robust distributed computing platform enables enhanced performance and allows for scalability. The following diagram offers a high-level conceptual view of the tiers. For a detailed description of this diagram, see Chapter 3, "Technical architecture".



RPOS's integration points into the retail enterprise

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



RPOS-related dataflow across the enterprise

Javadoc for RPOS

Javadoc is the tool from Sun Microsystems that generates API documentation in HTML format. Retek provides Javadoc documentation generated from RPOS code. Click the HTML file named 'index' in the applicable Javadoc folder to open the Javadoc.

Where you can find more information

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

- RPOS front-end documentation (for example, the RPOS User Guide and online help)
- RPOS Installation Guide
- Store Inventory Management (SIM) product documentation
- Retek Merchandising System (RMS) product documentation

Chapter 2 – Backend system administration and configuration

This chapter of the operations guide is intended for administrators who provide support and monitor the running system.

The content in this chapter is not procedural, but is meant to provide descriptive overviews of the key system parameters.

Supported Retek products

This version of RPOS is compatible with the following:

- SIM 11.0
- RMS 11.x
- RPM 11.0

Supported environments

For information about requirements for RPOS's client, application server, and database server, see the RPOS Installation Guide. Note that the compiler vendor depends upon the deployed hardware.

Configuration (.cfg) files

Some configurations for RPOS are located in .cfg files. The key system parameters contained in these file are described in this section.

Note that within these .cfg files (and thus in some of the examples from those files below), a # sign that precedes a value in the file signifies that what follows is a comment and is not being utilized as a setting.

Some settings in the .cfg files are configurable. Thus, when retailers install RPOS into an environment, they must update these values to their specific settings. Configuration files can be edited through a text editor or the Application Builder tool.

JDBC configuration file (jdbc.cfg)

This file delineates how the system uses the persistence layer. Key RPOS-related values within the file are shown below. Note that some values in the file may be intended for development purposes only or be related to another product (SIM, for example).

Database configuration

The configuration in this file instructs the system about the database in which the RPOS tables reside. Note that a retailer using a specific database comments out any other database parameters in the configuration file. Under normal circumstances, the driver value should *not* change. The retailer establishes the machine name, the database name, the user name, and the password. The pool size pertains to the number of available database connections that the retailer intends to keep available in the pool. The application automatically adjusts the pool size as needed. JDBC_VERBOSE setting is for debugging purposes only. If a database connection is unused for a certain amount of time, it stops automatically. The lock setting describes how long the system attempts to procure a lock for the user before giving up and returning a message that the lock could not be attained.

For example:

```
# Oracle configuration
DATABASE=ORACLE
DRIVER=oracle.jdbc.driver.OracleDriver
URL=jdbc:oracle:thin:@<RPOSDBMachineName>:1521:<RPOSDBName>
POOL_INITIAL_SIZE=5
USER_NAME=<RPOSUserId>
PASSWORD=<password>
JDBC_VERBOSE=false
#Time in seconds to keep unused connections (3600 = 1 hour)
CONNECTION_EXPIRATION=3600
#Time in seconds to wait on database locks
LOCK_WAIT=5
```

Specific data access objects (DAO) implementations

These settings relate to data access-related information. The values allow the retailer to customize what class is used for data access for a given service. The settings in this section are (with few exceptions) associated with Oracle. The values should not to be changed if the system was purchased with RMS, which must use Oracle. A retailer doing custom work, however, can write an implementation that works for whatever database is being used and change the values in this section accordingly.

For example:

```
#Specific DAO Implementations
COLLECTION_DAO=com.retek.iso.cs.dataaccess.artsoracle.dao.CollectionOracleDAO
COMMONTRANSACTION_DAO=com.retek.iso.cs.dataaccess.artsoracle.dao.CommonTransactionOracleDAO
CUSTOMER_DAO=com.retek.iso.cs.dataaccess.artsoracle.dao.CustomerOracleDAO
EMPAPPFORM_DAO=com.retek.iso.cs.dataaccess.artsoracle.dao.EmpAppFormOracleDAO
```

```
EMPFORM_DAO=com.retek.iso.cs.dataaccess.artsoracle.dao.EmpFormOracle
DAO

EMPLOYEE_DAO=com.retek.iso.cs.dataaccess.artsoracle.dao.EmployeeOrac
leDAO

EMPLOYEEAUTHINFO_DAO=com.retek.iso.cs.dataaccess.artsoracle.dao.Empl
oyeeAuthInfoOracleDAO

EMPLOYEETIMECARD_DAO=com.retek.iso.cs.dataaccess.artsoracle.dao.Empl
oyeeTimecardOracleDAO

EMPRESOURCE_DAO=com.retek.iso.cs.dataaccess.artsoracle.dao.EmpResour
ceOracleDAO

EMPSALE_DAO=com.retek.iso.cs.dataaccess.artsoracle.dao.EmpSaleOracle
DAO

EOD_DAO=com.retek.iso.cs.dataaccess.artsoracle.dao.EodOracleDAO
```

Network configuration file (network.cfg)

Connectivity between the RPOS client and the middle tier is achieved via a Remote Naming Service (RNS), which provides the RPOS client with the necessary IP address and port to access the RPOS container. This configuration file includes parameters related to the naming service of Integrated Store Operations (ISO). The client sets the IP address to the server on which the code is running. The port number (3000 in the example below) can be left as the default, or it can be changed.

Note: If the retailer changes the port number, the equivalent value must also be changed in RNS-related .bat/.sh files. The values must be kept in sync.

The MAXIMUM_CONNECT_ATTEMPTS value tells the system how many times it should attempt to connect before moving into offline mode, which prevents the user from continuing to use the system. The VERSION setting should not have to be changed.

```
#  This is a comma-delimited list of addresses to the master naming
#  service.

#  Example:  10.9.1.47:3000,10.9.1.47:4000,10.9.1.47:5000

#MASTER_NODE=<NamingServerIpAddress>:3000

MASTER_NODE=10.6.1.162:3000


#  This is the address to the naming service for a specific
#  application server.

#  Each application server will have a unique address.

#  The Node Monitor application  will bind to this.

#APPLICATION_NODE=<AppServerIpAddress>:3001

APPLICATION_NODE=10.6.1.162:3001


#  Maximum number of times to try to bind to a naming service before
#  throwing a DowntimeException "Unable to find Naming Service".

MAX_CONNECT_ATTEMPTS=2
```

```
#  Version control number of the naming service that any client
#  should look for.  This enables multiple versions to exist
#  simultaneously.

VERSION=1
```

Clientmaster configuration file (client_master.cfg)

This file contains information related specifically to the RPOS client. Note that some values in the file may be intended for development purposes only or be related to another product (a SIM product, for example).

Initial class that is run

These parameters are associated with initial system processing. Under normal operating conditions, these parameters should *not* be changed.

For example:

```
#  This is always the initial class that is run.  This class could
#  be used to update this file.

STARTUP_BOOT_STRAP=com.retek.iso.cr.appmgr.bootstrap.InitialBootStra
p

#  comma delimited class name that need to be executed before any
#  applications are loaded.

BOOT_STRAP=com.retek.iso.cr.appmgr.bootstrap.UpdateBootStrap,com.ret
ek.iso.cr.appmgr.bootstrap.ClientServicesBootStrap,com.retek.iso.cr.
appmgr.bootstrap.MerchandiseBootStrap,com.retek.iso.cr.appmgr.bootst
rap.CorpMsgBootStrap,com.retek.iso.cs.sos.SOSBootStrap,com.retek.iso
.cr.appmgr.bootstrap.GUIBootStrap,com.retek.iso.cs.item.ItemBootStra
p,com.retek.iso.cs.pricing.PromotionBootStrap,com.retek.iso.cs.emplo
yee.EmployeeBootStrap
```

Class names that represent daemons

Where bootstraps run at the start of the program, daemons are threads that run continually while the client is running. Retailers may never need to change this setting unless they are adding additional daemons to the system.

For example:

```
# comma delimited class name that represent deamons. This is run

# after the bootstrap is completed.

DAEMON=com.retek.iso.cr.appmgr.daemon.UpdateDaemon,com.retek.iso.cs.
merchandise.MerchandiseDaemon,com.retek.iso.cr.appmgr.daemon.Multica
stReaderDaemon,com.retek.iso.cr.appmgr.daemon.FileTransferDaemon
```

Message senders and message receivers

The parameters below are designed for RPOS's JMS functionality. RPOS provides the ability for registers and servers to send messages to topics and queues on a Java Messaging Service (JMS) server. The messaging framework allows the clients to receive messages from topics and queues on a JMS server. RPOS uses messaging via JMS to perform tasks such as updating the client. For example, a price update during the day could be pushed out through messaging functionality from a price update sender to a price update receiver.

The parameters below provide the system with the names of the message receivers and the message senders, either of which may be preconfigured with the system or created by the retailer. A retailer that created a message sender or a message receiver would have to configure the values in this portion of the file.

For more information regarding messaging, see "Chapter 6 – Messaging framework".

For example:

```
# Messaging - started after daemons have been started.

# comma delimited lists of MessageReceiver and MessageSender
subclasses

# to be run by the MessagingManager.

MESSAGE_RECEIVERS=com.retek.iso.cr.messaging.CorporateMessageReceive
r,com.retek.iso.cr.messaging.ConfigurationChangeReceiver,com.retek.i
so.cs.item.PriceUpdateReceiver,com.retek.iso.cr.messaging.ClearExcep
tionReceiver

MESSAGE_SENDERS=com.retek.iso.cr.messaging.InformationMessageSender,
com.retek.iso.cr.messaging.ExceptionMessageSender
```

Minute delay for JMS messaging

The parameters below determine the intervals the system uses to check for the existence of messages on the JMS queue(s) and the intervals the system uses to send messages to the JMS queue(s). The values are in minutes and are according to message type. In one of the values below, for example, the system checks for messages of the message type 'corporate receiver' every ten minutes.

For example:

```
EXCEPTION_SENDER_INTERVAL=5

CORP_MESSAGE_RECEIVER_INTERVAL=10

CONFIG_CHANGE_RECEIVER_INTERVAL=1

PRICE_UPDATE_RECEIVER_INTERVAL=5
```

End of day or end of session download waiting times

After an ‘end of day’ (EOD) or an ‘end of session’ (EOS) has been completed, the system’s applicable daemon waits this many minutes before attempting to update its item file(s) or other files set up on the retailer’s headquarters application server.

For example:

```
# maximum number of minutes to wait after eod/eos before download
code updates

UPDATE_DOWNLOAD_WAIT=360
```

After an ‘end of day’ (EOD) or an ‘end of session’ (EOS) has been completed, the system’s applicable daemon waits this many minutes before attempting to update its merchandise media.

For example:

```
# maximum number of minutes to wait after eod/eos before download
merchandise

MERCHANDISE_DOWNLOAD_WAIT=360
```

Minimum time for data refresh

The system does not update its files unless this number of hours has passed. Because of these values, if a client is frequently restarted, the system considers its data to continue to be valid.

For example:

```
#Number of hours before the items file will be updated, since the
last update

ITEMS_DOWNLOAD_HOURS=12

#Number of hours before the employees file will be updated, since
the last update

EMPLOYEES_DOWNLOAD_HOURS=12

#Number of hours before the promotions files will be updated, since
the last update

PROMOTIONS_DOWNLOAD_HOURS=12
```

Peer-to-peer communication

Over the local network, RPOS uses a multicast method of communication in its peer-to-peer client communications. Through multicast, a source host sends a message to a group of destination hosts. In order to facilitate peer-to-peer processing (that is, for the clients to be connected to one another), the values below must be the same for every RPOS client.

For more information about peer-to-peer processing, see the section, ‘Key components in the client architecture’ in “Chapter 3 – Technical architecture”.

For example:

```
#peer to peer parms, ex 224.3.3.15; 2001; 60000

MULTICAST_ADDRESS=224.3.3.15

MULTICAST_PORT=2001

MULTICAST_RATE=60000
```

Password required

The parameter below determines whether or not the user, after signing in, must enter a password to gain access to the client.

For example:

```
# Require the employee to also enter a password after providing a signon
IS_PASSWORD_REQUIRED=false
```

Cashier session and ‘go home’

Once the user logs in, he or she can start a cashier session. The ‘go home’ in RPOS is the starting screen where the user logs in. This value determines whether the user has to log in once the user performs a ‘go home’. Retailers can use this setting as a way to make a new cashier sign in after the previous cashier has signed off.

For example:

```
#Can the user continue after each Go Home without logging in
CASHIER_SESSION=false
```

Peers

These settings, described below, pertain to classes that are instrumental to the system’s peer-to-peer communication among the clients. A retailer adding new components of peer-to-peer functionality would have to add any applicable additional class names here.

Park

The ‘park’ parameter helps enable a user to ‘park’ a transaction. For example, suppose a user is in the middle of a transaction on one client and suspends it. The user could go to another client and, because of the system’s peer-to-peer functionality, recall the same transaction. When instructed to recall a client that does not currently have the transaction, it goes to the other clients to determine if they do. Suspend/recall functionality thus works across registers.

For example:

```
PEER.park=com.retek.iso.cr.park.ParkRMIPeerImpl
```

Employee

If a client does not have an employee, this parameter helps enable the client go to other clients, through the system’s peer-to-peer functionality, to find the employee data.

For example:

```
PEER.employee=com.retek.iso.cs.employee.CMSEmployeeRMIPeerImpl
```

Update

Note: The value below pertains to non-bootstrap-related updates only.

When updates are downloaded, only one client performs the download. Through the peer-to-peer functionality, the updates are then shared with the other clients.

For example:

```
PEER.update=com.retek.iso.cr.download.update.UpdateRMIPeerImpl
```

Persist

Within RPOS, Java objects can be stored as files. Through peer-to-peer functionality, clients can share this data.

For example:

```
PEER.persist=com.retek.iso.cr.persist.PersistRMIPeerImpl
```

Peer timeout

These parameters represent in milliseconds the amount of time a peer client waits before it stops trying to communicate with another peer client.

For example:

```
# Peer timeout - number of milliseconds that the peer will wait
before a timeout

PEER_TIMEOUT=2000
```

Logging

This parameter contains logging information related specifically to the RPOS client. The retailer can specify the log filename. The system's default uses the directory specified, and separate log files can exist for unique clients. The LOGGING_PAUSE value tells the system how long to wait before writing data to the log file.

For example:

```
#logging

LOGGING_IMPL=com.retek.iso.cr.logging.LoggingFileServices

#  uncommenting this line will use this as the client log filename
#  rather than the name that includes the timestamp

#LOGGING_FILE_NAME=..\\log\\client.log

LOGGING_LEVEL=4

LOGGING_PAUSE=5000

LOGGING_SYSTEM_OUT=true

LOGGING_SYSTEM_ERR=true
```

List of resource bundles

Resource bundles are Java files related to the internationalization process. This entry specs the system's default resource bundles. Language-specific versions are handled automatically based on the configured user's locale. These values should *not* have to be changed even when the system is offered in different languages.

For example:

```
   # Comma-delimited list of classes of resource bundles
MESSAGE_BUNDLE=com.retek.iso.cs.util.MessageBundle,com.retek.iso.cs.uti
l.MnemonicMessageBundle,com.retek.iso.cs.util.RuleMessageBundle,com.ret
ek.iso.cs.util.ConfigMessageBundle,com.retek.iso.cs.util.TAConfigMessag
eBundle,com.retek.iso.cs.util.TAMessageBundle,com.retek.iso.cs.util.Mas
kResourceBundle
```

Look and feel of the client

These parameters allow the retailer to customize the appearance of the application on the PC. However, Retek does *not* recommend that these values be changed.

For example:

```
   #  Attempt to use one of the following Look And Feels from LF1 to
   #  LFn.
   LOOK_FEEL_KEY=LF1,LF2,LF3,LF4,LF5
   LF1=com.retek.iso.cr.swing.plaf.metal.ConfigurableMetalLookAndFeel
   LF2=com.apple.mrj.swing.MacLookAndFeel
   LF3=com.sun.java.swing.plaf.windows.WindowsLookAndFeel
   LF4=javax.swing.plaf.metal.MetalLookAndFeel
   LF5=com.sun.java.swing.plaf.motif.MotifLookAndFeel

   # The direction the toolbar and appbar will show up
   #     0 - Bottom Right
   #     1 - Bottom Left
   #     2 - Upper Right
   #     3 - Upper Left
   USER_PREFERENCES.ANCHOR=0
   # Show a splash
   SHOW_SPLASH=FALSE
```

```
#  The milliseconds delay to wait before enabling a button on a
#  newly displayed applet

# Suggested value: 200 (two tenths of a second)

BUTTON_DELAY=0

#  whether the button text should be displayed with HTML.  Useful
#  for word-wrapping.

FORMAT_BUTTONS_WITH_HTML=false
```

Initial drawer fund

This parameter instructs the system to prompt the user, at the start of the day, to input the amount that resides within the cash drawer. The user can enter an amount but does not have to.

For example:

```
# use initial drawer fund?

USE_INITIAL_DRAWER_FUND=YES
```

Client services loaded by bootstraps

These parameters are associated with the initial system processing of client services. Under normal operating conditions, these parameters should *not* have to be changed.

For example:

```
#Client Services loaded by bootstraps

SERVICES_LIST=AUTH_SRVC,CUSTOMER_SRVC,CREDIT_AUTH_SRVC,EMPLOYEE_SRVC
,EMPLOYEERESOURCE_SRVC,FILETRANSFER_SRVC,FORM_SRVC,GOAL_SRVC,ITEM_SR
VC,LAYAWAY_SRVC,MERCHANDISE_SRVC,PROMOTION_SRVC,READINGS_SRVC,REDEEM
ABLE_SRVC,REGISTER_SRVC,ROLE_SRVC,SCHEDULE_SRVC,STORE_SRVC,TAX_SRVC,
TIMECARD_SRVC,TRANSACTIONEOD_SRVC,TRANSACTIONSOS_SRVC,TXN_NUMBER_SRV
C,TXN_POS_SRVC,TXN_POSTER_SRVC,VALUEADDEDTAX_SRVC


AUTH_SRVC=com.retek.iso.cs.auth.CMSAuthClientServices

CUSTOMER_SRVC=com.retek.iso.cs.customer.CMSCustomerClientServices

CREDIT_AUTH_SRVC=com.retek.iso.cs.authorization.bankcard.CMSCreditAu
thClientServices

EMPLOYEE_SRVC=com.retek.iso.cs.employee.CMSEmployeeClientServices

EMPLOYEERESOURCE_SRVC=com.retek.iso.cs.scheduling.CMSEmployeeResourc
eClientServices

FILETRANSFER_SRVC=com.retek.iso.cs.filetransfer.CMSFileTransferClien
tServices

FORM_SRVC=com.retek.iso.cs.forms.CMSFormClientServices

GOAL_SRVC=com.retek.iso.cs.goaling.CMSGoalingClientServices

ITEM_SRVC=com.retek.iso.cs.item.CMSItemClientServices

LAYAWAY_SRVC=com.retek.iso.cs.layaway.CMSLayawayClientServices

MERCHANDISE_SRVC=com.retek.iso.cs.merchandise.CMSMerchandiseClientSe
rvices
```

```
PROMOTION_SRVC=com.retek.iso.cs.pricing.CMSPromotionClientServices
READINGS_SRVC=com.retek.iso.cs.readings.CMSReadingsClientServices
REDEEMABLE_SRVC=com.retek.iso.cs.payment.CMSRedeemableClientServices
REGISTER_SRVC=com.retek.iso.cs.register.CMSRegisterClientServices
ROLE_SRVC=com.retek.iso.cs.scheduling.CMSRoleClientServices
SCHEDULE_SRVC=com.retek.iso.cs.scheduling.CMSScheduleClientServices
STORE_SRVC=com.retek.iso.cs.store.CMSStoreClientServices
TAX_SRVC=com.retek.iso.cs.tax.CMSTaxClientServices
TIMECARD_SRVC=com.retek.iso.cs.timecard.CMSTimecardClientServices
TRANSACTIONEOD_SRVC=com.retek.iso.cs.eod.CMSTransactionEODClientServices
TRANSACTIONSOS_SRVC=com.retek.iso.cs.sos.CMSTransactionSOSClientServices
TXN_POS_SRVC=com.retek.iso.cs.pos.CMSTransactionPOSClientServices
TXN_POSTER_SRVC=com.retek.iso.cs.txnposter.CMSTxnPosterClientServices
TXN_NUMBER_SRVC=com.retek.iso.cs.txnnumber.CMSTransactionNumberClientServices
VALUEADDEDTAX_SRVC=com.retek.iso.cs.tax.CMSValueAddedTaxClientServices
```

Logging information

Default location of client and server log files

Unix

Log files related to the client are located in the following directory:

```
clientUnix\retek\RPOS\files\prod\log
```

Log files related to the server are located in the following directory:

```
serverUnix\retek\RPOS\files\prod\log
```

Windows

Log files related to the client are located in the following directory:

```
clientWindows\retek\RPOS\prod\log
```

Log files related to the server are located in the following directory:

```
serverWindows\retek\RPOS\files\prod\log
```

Logging levels established in configuration files (.cfg)

Logging levels can be established configuration files (such as item.log). This level of logging can be helpful when troubleshooting specific parts of the application. For example, if the application is experiencing issues in a specific area, the logger can set to a higher degree of granularity. For example, the stockcount.cfg file contains the following entry:

```
# Logging
LOGGING_IMPL=com.retek.iso.cr.logging.LoggingFileServices
LOGGING_FILE_NAME=../log/item.log
LOGGING_LEVEL=4
LOGGING_PAUSE=5000
LOGGING_SYSTEM_OUT=true
LOGGING_SYSTEM_ERR=true
```

Configuring JPOS peripherals with retek_jpos.xml and jpos_peripherals.cfg

The following high-level steps provide an outline of what is necessary to configure peripherals for RPOS. The actual installation of peripherals can be a highly involved because of the wide variety of driver vendors (such as IBM), device-specific driver versions, and so on.

1 Install the JPOS-compliant drivers specific to the peripherals.

2 In `Javapos/retek_jpos.xml`, set up the Javapos definition of peripheral for the driver that was installed. Configure the applicable values in the file for the device-level specification. Formats defined for different devices are defined per the JPOS standard.

3 Configure the `jpos_peripherals.cfg` so that the supported JPOS device type maps to the name used to define the device in the `retek_jpos.xml`.

Exception handling

The primary types of exceptions within the RPOS system include the following:

- com.retek.iso.cr.rules.BusinessRuleException
  This exception indicates that a business rule has been violated.
- com.retek.iso.cr.appmgr.ApplicationException
  This exception is used to wrap lower system-level exceptions that occur throughout the framework. This class should be used to specify what the application needs to handle this exception with business logic, allowing lower level exceptions that occur to fall through and be handled differently.
- com.retek.iso.cs.dataaccess.DAOException
  A DAOException is an exception thrown by the DAO. When some code has caught an exception in a DAO, it should throw a DAOException upward as a result. This class allows the original exception to continue to be accessible.

Java Virtual Machine (JVM) options

Any JVM modifier (that is, any that can be specified in the 'Java' program) can be specified in the POSMainContainer.xml file.

For example, if out of memory issues occur because of the amount of data involved during runtime, the JVM setting can be adjusted in the section of the RSSMainContainer.xml file shown below. Note that in the example below the 128m stands for 128 megabytes, the value that would need to be increased.

For example:

```
<jvmLineArgs length="2">
            <java.lang.String>-Xms4m</java.lang.String>
            <java.lang.String>-Xmx128m</java.lang.String>
        </jvmLineArgs>
```

Pos.cfg

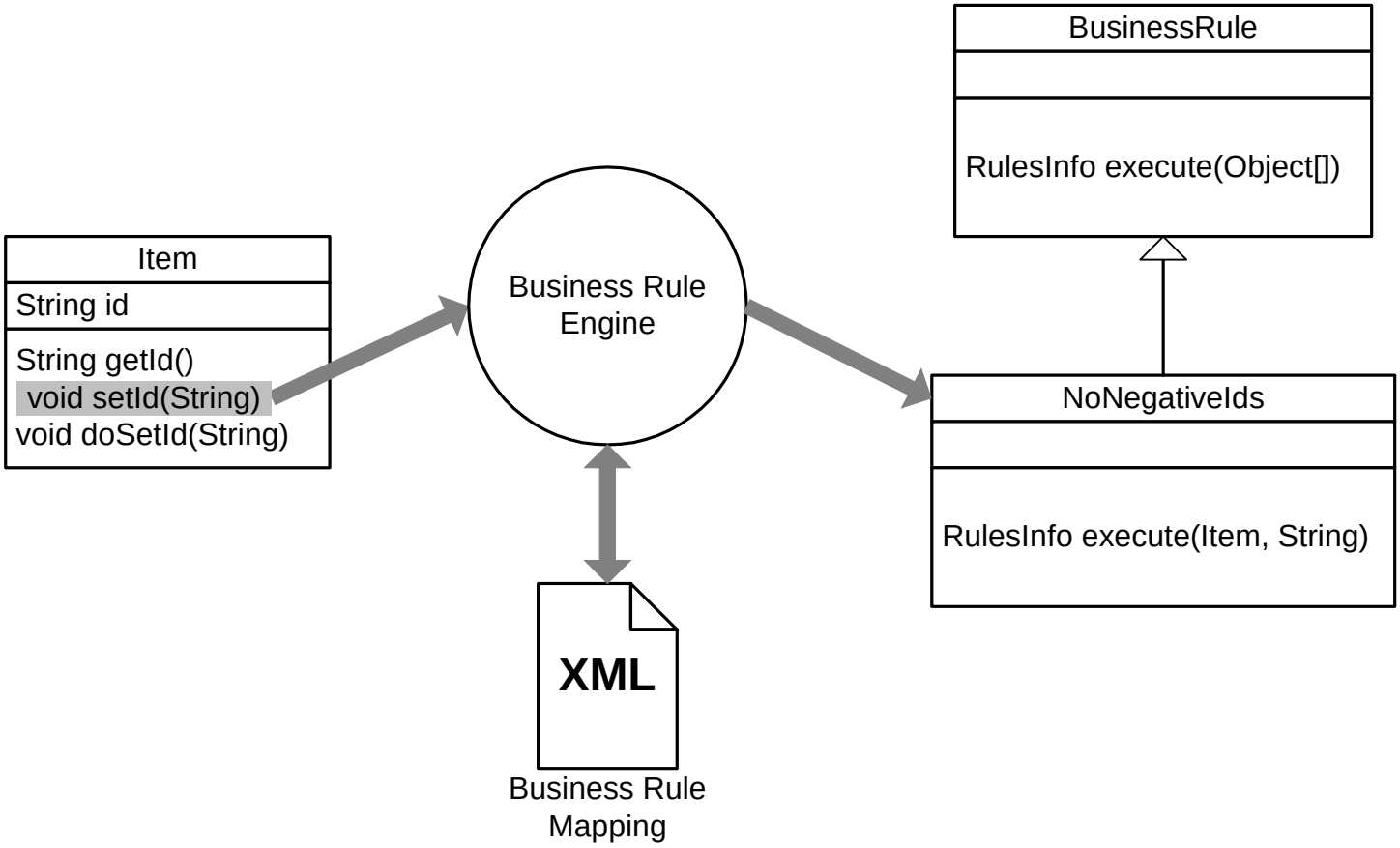
This file should not have to be changed. The file contains transaction types that are supported throughout the system's code.

Business rules configuration through rules.xml

Business analysts can use RPOS's business rule engine to modify the behavior of the RPOS system. Business objects enforce and encapsulate business rules keeping objects from being corrupted by faulty user-interface code. Business rules are a quick way for a developer to change the behavior of a business object or its process without adding to core behavior. For more information about business objects, see "Chapter 3 – Technical architecture".

Business rules are pluggable which allows them to be modified separately from the core code. They can be added today and removed tomorrow. Once a rule written, an analyst can add or remove the rule without the use of a developer.

The following diagram and the bullet points that follow the diagram provide an illustration of the rules engine and the role that rules.xml plays in relation to it.



Business rule engine and configuration

Business object ‘set’ methods call the rule engine.

- Application Builder is used to build rules mapping.
- The rule engine maps methods to rules.
- Rules execute and return success / fail.
- Information includes message to users.

For example, a portion of rules.xml is shown below:

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <CMS_OBJECT>
- <RULE_REPOSITORY>
  <ruleExecutionLoggingRequested>false</ruleExecutionLoggingRequested>
  <ruleFailureLoggingRequested>false</ruleFailureLoggingRequested>
- <RULE_ASSIGNMENT name="ADDITIONAL_EMP_INFO.isShowable">
  <ruleAssignmentRuleClass
index="0">com.retek.iso.cs.rules.employee.AdditionalEmpInfoButtonShows</rule
AssignmentRuleClass>
  </RULE_ASSIGNMENT>
- <RULE_ASSIGNMENT name="ADDITIONAL_EMP_INFO_PREV.isShowable">
  <ruleAssignmentRuleClass
index="0">com.retek.iso.cs.rules.employee.StandardEmpInfoButtonShows</ruleAs
signmentRuleClass>
  </RULE_ASSIGNMENT>
- <RULE_ASSIGNMENT name="ADD_BENEFIT.isShowable">
  <ruleAssignmentRuleClass
index="0">com.retek.iso.cs.rules.timecard.EmployeeMayNotModOwnTimecard</rule
AssignmentRuleClass>
  <ruleAssignmentRuleClass
index="1">com.retek.iso.cs.rules.timecard.EmployeeMayNotModOtherStoresTimeca
rd</ruleAssignmentRuleClass>
  </RULE_ASSIGNMENT>
- <RULE_ASSIGNMENT
name="com.retek.iso.cs.pos.CMSLayawayLineItem.setQuantity">
  <ruleAssignmentRuleClass
index="0">com.retek.iso.cs.rules.item.MiscItemsNotEligibleForQtyChange</rule
AssignmentRuleClass>
  </RULE_ASSIGNMENT>
- <RULE_ASSIGNMENT
name="com.retek.iso.cs.pos.CMSReturnLineItem.adjustMarkdownAmount">
  <ruleAssignmentRuleClass
index="0">com.retek.iso.cs.rules.lineitem.LineItemMarkdownIsValid</ruleAssig
nmentRuleClass>
  </RULE_ASSIGNMENT>
- <RULE_ASSIGNMENT
name="com.retek.iso.cs.pos.CMSReturnLineItem.adjustUnitPrice">
```

```
  <ruleAssignmentRuleClass
index="0">com.retek.iso.cs.rules.lineitem.AdjustManualUnitPriceIsValid</rule
AssignmentRuleClass>
- <RULE_ASSIGNMENT
name="com.retek.iso.cs.pos.CMSReturnLineItemDetail.clearManualTaxAmount">
  <ruleAssignmentRuleClass
index="0">com.retek.iso.cs.rules.lineitem.TaxAmountCannotBeClearedOnReturnLi
neDetail</ruleAssignmentRuleClass>
  </RULE_ASSIGNMENT>
  </RULE_ASSIGNMENT>
- <RULE_ASSIGNMENT
name="com.retek.iso.cs.transfer.CMSTransferOut.testIsVoidable">
  <ruleAssignmentRuleClass
index="0">com.retek.iso.cs.rules.transaction.TransferOutCannotBeVoidedIfComp
leted</ruleAssignmentRuleClass>
  </RULE_ASSIGNMENT>
  </RULE_REPOSITORY>
  </CMS_OBJECT>
```

Chapter 3 – Technical architecture

This chapter describes the overall software architecture for RPOS, offering a high-level discussion of the general structure of the system, including the various layers of Java code. This information is valuable in the following scenarios, among others:

- When the retailer wishes to take advantage of RPOS's extensible capabilities and write its own code to fit into the RPOS system.
- When the retailer wishes to implement the system for various databases (Oracle, DB2, and so on).

For those who are less familiar with Java terminology, a description of RPOS-related Java terms and standards is provided for your reference at the end of this chapter.

Overview

RPOS and Integrated Store Operations (ISO)

ISO is a group of in-store operations applications that will use and share common business logic, configuration utilities and technology infrastructure. The applications are proven to simplify store operations by improving customer service and decreasing costs. RPOS is an application on the ISO platform.

Advantages of the architecture

Scalability

RPOS's robust distributed computing platform enables enhanced performance and allows for scalability. Hardware and software can be added to meet retailer requirements for each of the tiers. When component services no longer support the number of user-defined requests, horizontal scaling is necessary to add additional software services. RPOS is comprised of component services that represent a retailer's business requirements. These services are loaded (or unloaded) according to user-defined parameters (scaling to meet the user's needs). For example, if a single service can handle 50 concurrent requests, 2 services can handle 100 concurrent requests, 3 can handle 150, and so on.

Vertical scaling is necessary when hardware can no longer support additional services; the retailer can add additional hardware or upgrade existing hardware for vertical scaling. For example, a retailer might add a processor, upgrade processors, add more memory, or add new hardware to the application server farm.

High availability

RPOS's availability is designed to increase based on customer demands. It has no single-point-of-failure, software or hardware, and while it is designed for server clusters, clusters are not required. Middleware is designed to run in a clustered environment or on a low-cost blade server.

Flexibility

RPOS supports multiple network topologies through configuration rather than through extensive code changes. RPOS supports multiple database interfaces. RPOS's Java-based tiers facilitate operating system (OS) independence while employing existing hardware. For the retailer, data placement can be based on business requirements rather than on technical limitations. The code has been written to run wherever Java is run. The same object model is on all devices within the platform, regardless of the operating system being used on individual devices.

Fault toleration

RPOS continues to operate in the event of system failure. Each POS terminal within the local area network (LAN) is mirrored and has the ability to operate independent of the LAN and wide-area network (WAN). In the event of a failure within the LAN or WAN, POS terminals switch to offline mode without user intervention and continue to function.

Cost effectiveness

RPOS uses open source market-proven technology. Object-oriented design increases reusability for faster development and deployment. The reuse of business objects and function allows for faster integration to enterprise subsystems. N-tier architecture has become an industry standard. Multi-tiered physically distributed architecture extends the life of the system.

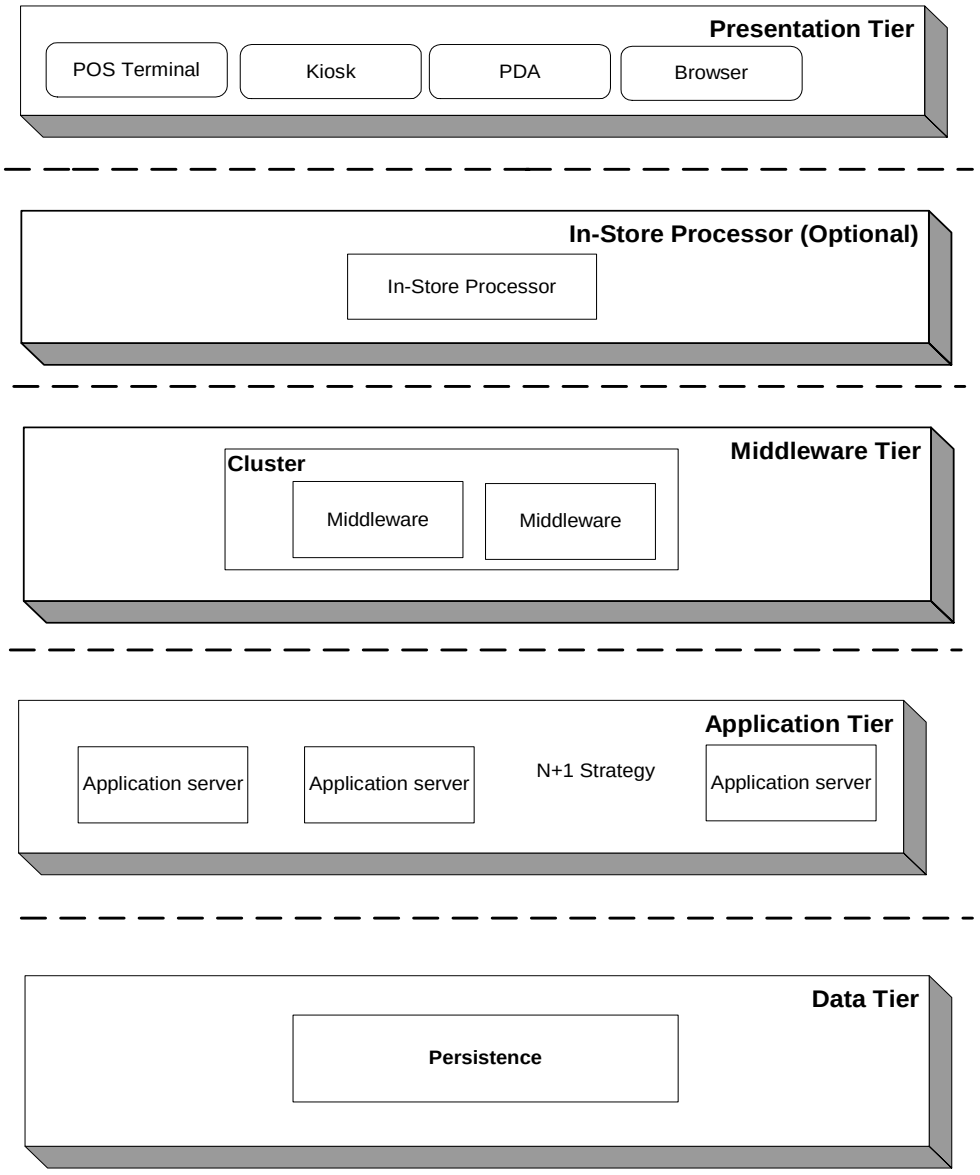
Manageability

The n-tier architecture of RPOS allows for the encapsulation of business logic, shielding the client from the complexity of the back-end system. Any given tier need not be concerned with the internal functional tasks of any other tier.

RPOS provides a retailer with the ability to remotely manage, debug, and tune application servers. POS terminals within the platform self-update allowing for quick and easily managed updates.

A high-level view of the tiered model

The following diagram, together with the explanations that follow, offer a high-level conceptual view of the tiers and their responsibilities within the architecture.



Conceptual view of the tiered model

Presentation/client tier

This tier handles the presentation of the application, including its user interface. The presentation tier consists of the store-level clients on which RPOS runs. The presentation tier allows for retailers to implement multi-channel presentations without modifying the architecture's backend. The GUI is responsible for presenting data to the user and for receiving data directly from the user through the 'front end'. The presentation tier only interacts with the middle tier (as opposed to the database tier). Various new interfaces can be constructed using the same core business objects.

Business process flow from a user's point of view

The presentation tier drives business process flow from the user's point of view. Sales associates and store managers employ POS terminals to ring a sale, manage a store, and so on. Customers and sales associates can use kiosks and PDAs to create a bridal registry, scan items, and suspend transactions for retrieval at another POS device. With a browser, users can make purchases online while sitting at home. In all instances, business flow is driven by the user.

Rich client

To facilitate the demands and complexity of point-of-sale demands, the RPOS front end facilitates robust client-side processing. The POS terminal interface was developed using Swing, which is a toolkit for creating rich graphical user interfaces (GUIs) in Java applications. Caching is utilized for increased performance. Abstract client services provide transparent access to network implementations.

Peripherals

The client interacts with POS peripherals using jPOS, an opensource Java-based financial transaction library/framework. RPOS also supports some peripherals using standards other than jPOS. The system can be configured to support jPOS-compliant peripherals that include the following:

- Printers
- Scanners
- Cash drawer
- Line display

Note: Magnetic strip readers (MSR) can be supported via methods other than JPOS.

- MSR

The client framework is designed as a touchscreen and keyboard-driven Java application. There is no need to attach a mouse to drive the application.

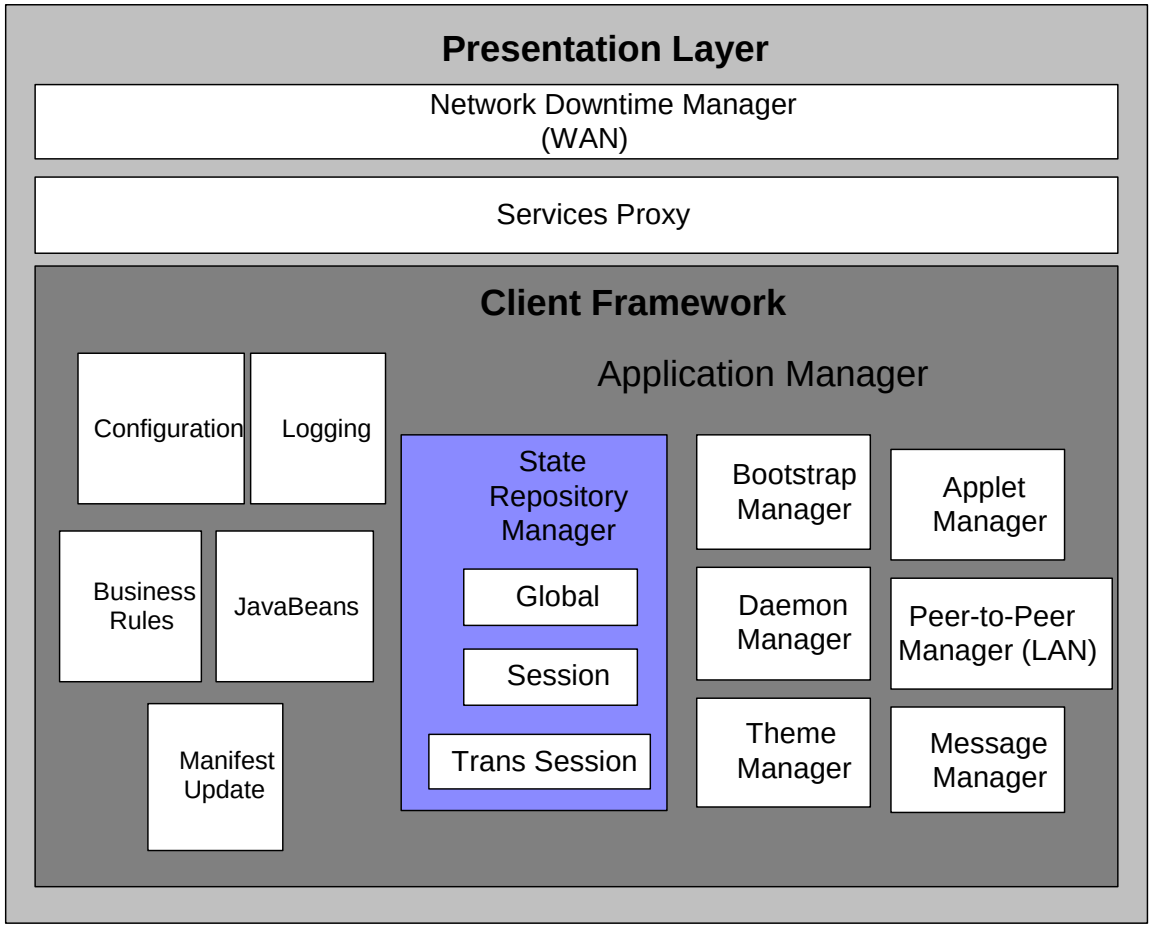
For information about the configuration of peripherals, see "Chapter 2 – Backend system administration and configuration".

Pluggable business rules

The client architecture includes pluggable business rules. For more information about business rules, see "Chapter 2 – Backend system administration and configuration".

Key components in the client architecture

The diagram below illustrates key components within the client. Explanations of each item on the diagram follow it.

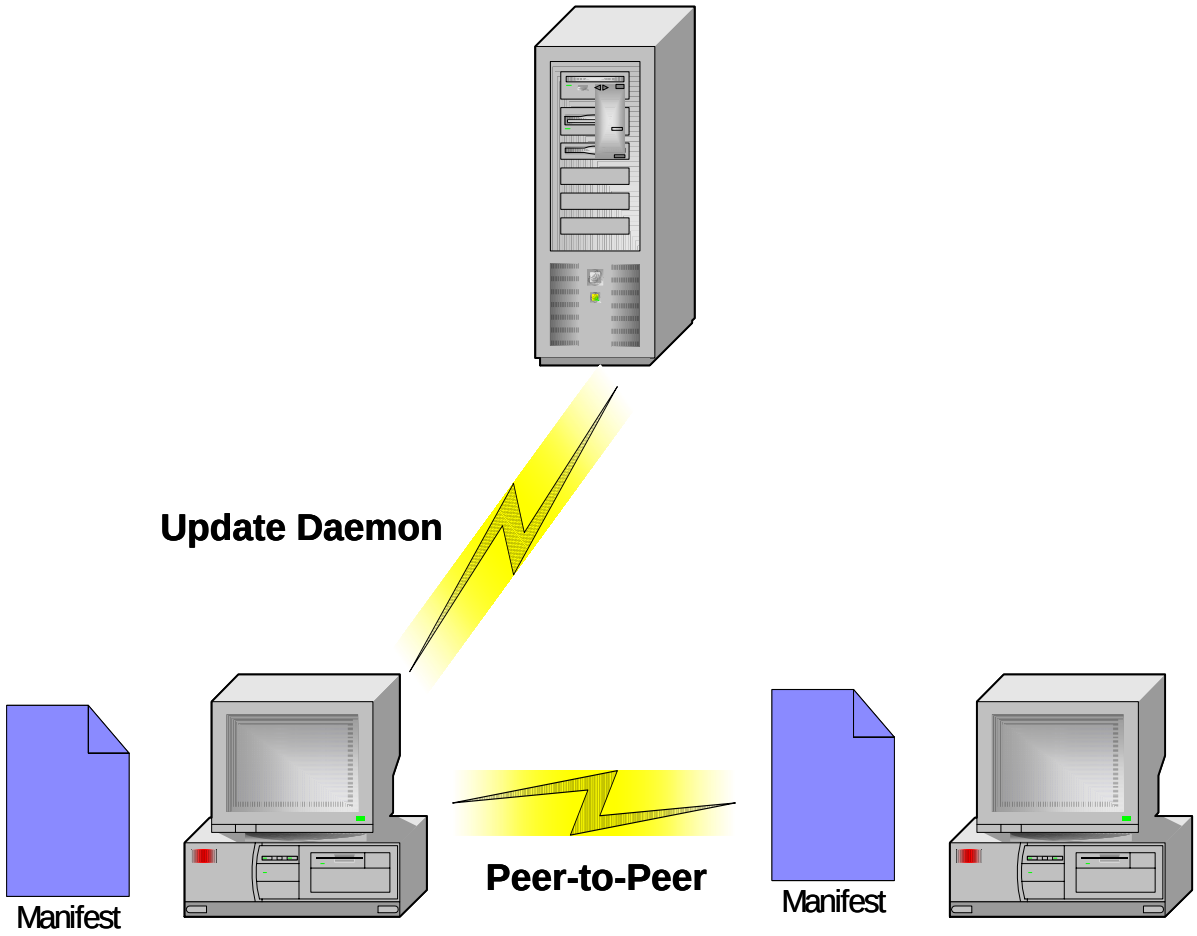


RPOS's key client architectural components

- State repository manager
  - Global repository
    The repository provides access to persisted objects. Security prevents application programmers from overwriting global objects. All client services' objects are stored in the repository for easy access. The system provides serialization option (to local storage).
  - Session repository
    This component provides the same functionality as global repository, but for transient objects. The system has relaxed security for application programmers. The repository is 'cleaned out' when a transaction is completed.
  - Trans session repository
    The new trans session state repository is intended for objects existing between operator sessions.

- Application manager
  The application manager provides a common platform for all client processes. Thus RPOS includes a common programming API interface into GUI development. Commonly used methods include the following:
  - `setTheme()`
  - `goBack()`
  - `goHome()`
  - `showApplet()`
  - `showErrorDlg()`
  - `showOptionDlg()`
  - `showMenu`
- Applet manager
  This component is responsible for managing the applets that form an application. All screens extend CMSApplet which contains all the hooks into the framework. The component manages the lifecycle of an applet and caches all applets for increased performance. Commonly used methods include the following:
  - `init()`
  - `start()`
  - `stop()`
  - `getVersion()`
  - `pageUp()`
  - `pageDown()`
  - `getScreenName()`
- Network downtime manager
  This component manages all client services. If any client service fails, the component sends all client services into ‘offline’ mode. In offline mode, the component monitors the network for an ‘up’ condition and reconnects all client services.
- Bootstrap manager
  This configuration-based component dynamically manages the start-up process. The component must extend BootStrap class to inherit base behavior.
- Daemon manager
  This configuration-based component dynamically manages all daemon threads. The component must extend Daemon class to inherit base behavior.
- Theme manager
  Themes are comprised of various images, fonts, and colors that determine the look and feel of RPOS. This component is incorporated into JavaBeans for global configuration. The component facilitates the retailer’s ability to change themes dynamically.

- Peer-to-peer manager
  This component, which uses a combination of TCP/UDP network protocols, is responsible for the following:
  - Suspend / recall transactions
  - Employee and item file updates
  - Manifest software updates
- Manifest update daemon
  This configurable component provides automatic updates to client software or files. Modes (wait, install, and remove) are provided for more control of the update process. As the diagram below illustrates. This component uses peer-to-peer communication for LAN distribution.



Manifest software deployment process

In-store processor (optional)

In-store processors act as additional application servers and allow for thinner clients. This tier is an optional tier in the ISO logical architecture. An in-store processor is not required in POS's logical architecture.

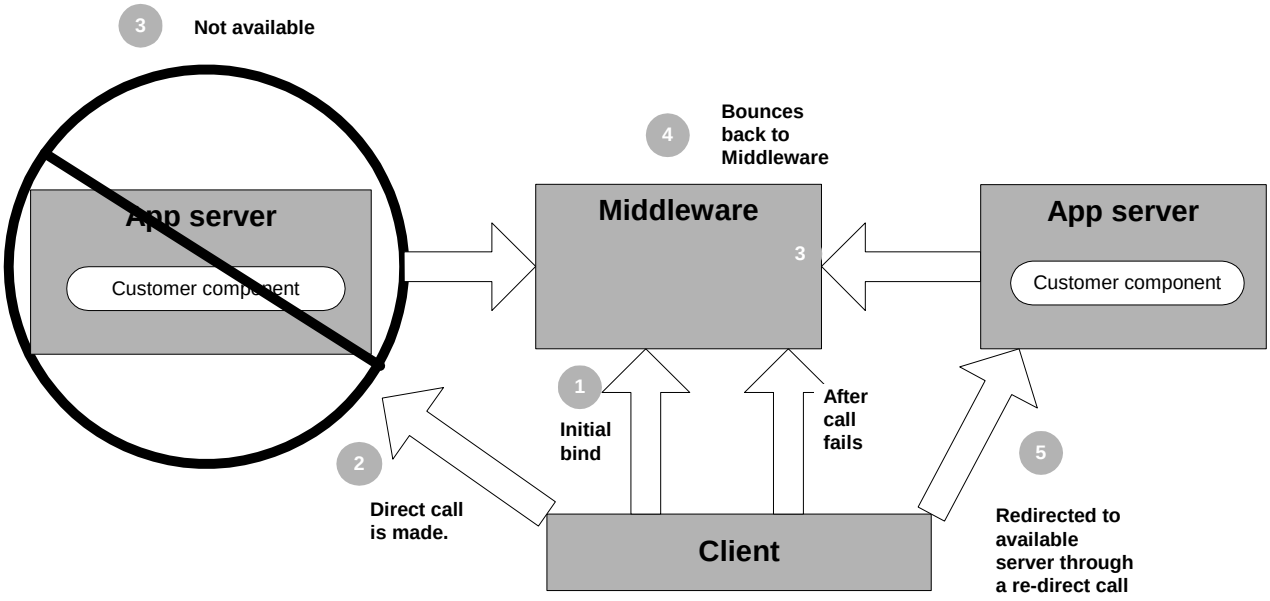
Middleware tier

Connectivity between the RPOS client and the middle tier is achieved via a Remote Naming Service (RNS), which provides the RPOS client with the necessary IP address and port. The remote naming service (RNS), or middleware tier of the RPOS platform, essentially serves as a communication bridge between the clients and the application server.

The RNS is a single server or a clustered configuration of servers sharing the same virtual IP address. The RNS provides the store-level clients on the presentation tier with references to POS services running on the application tier. The IP address of the RNS is all that is required for the clients.

The middleware tier handles high availability, fault tolerance, remote object binding or lookup, and load distribution. The middleware can reside at the application tier level in a non-production environment.

The following diagram describes how middleware directs traffic to available application servers.



Middleware directing traffic

Application tier

The application tier is where the application servers reside. All client requests are processed on the application servers. The application servers communicate directly with the database tier, process the data, and return the results to the clients.

The application tier provides vertical scalability and horizontal scalability for the presentation tier and employs an N+1 strategy for high availability. Application servers within this tier house remote components or POS services that can be accessed from anywhere on the network. Services can provide complicated processing, access to data, and interaction with another system that deals with a specific business function (for example, inventory or merchandise subsystems).

Data tier

The data tier is where the database servers and data reside. This tier, also called the persistence tier, is completely transparent to the client while providing data persistence. This tier can contain legacy applications such as inventory management and merchandising systems that provide behavior for current applications. This tier is configured based on the preferences and needs of a retailer's business.

RPOS object methodology

An object is a self-contained software entity that consists of both data and procedures to manipulate the data. The RPOS architecture uses streamlined object modeling based on the Coad Methodology. The Coad Methodology is a pure object approach in which system objects are designed around what the domain is, rather than around how to solve problems within a domain. Objects are easily extended to provide new domain-specific capabilities.

Business objects

RPOS business objects are separate from the user interface and database. In other words, presentation is separate from processing, which is separate from persistence. Separated business objects allow for new interfaces to be constructed using the same core business objects.

Business objects enforce and encapsulate business rules keeping objects from being corrupted by faulty user-interface code. Business rules are pluggable which allows them to be modified separate from the core code. Analysis, design, and implementation patterns ensure a highly extensible framework.

Distributed topology

One of RPOS's most significant advantages is its flexible distributed topology. RPOS offers complete location transparency because the location of data and/or services is based upon the retailer's business requirements, not upon technical limitations. RPOS's client server connection utilizes Remote Method Invocation (RMI). For RPOS, the use of RMI means that the application can take advantage of distributed objects; that is, the server can be geographically distributed, residing at a central location. Because the server does not have to be in the same store as the in-store clients, the clients log onto the server 'over the wire'.

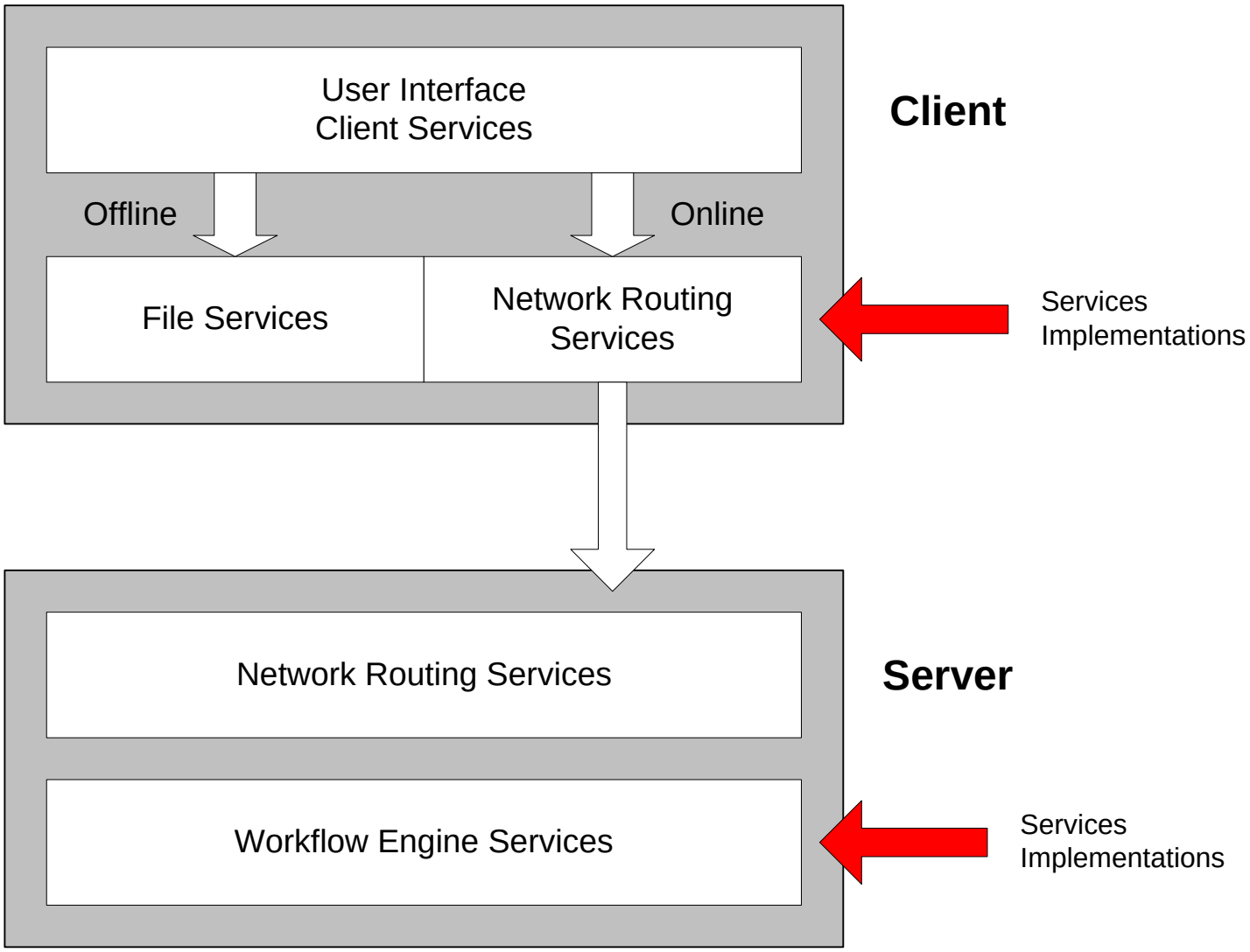
RPOS's RMI stub resides on the client side of the client/server relationship. The RMI stub contains no business logic but contains only enough code to effectively 'pass through' data. As far as the clients are concerned, all the processing occurs locally because they interact only with the stub, which is local. The GUI uses helper classes to 'talk' with the stub.

Connectivity between the RPOS client and the middle tier is achieved via a Remote Naming Service (RNS), which provides the RPOS client with the necessary IP address and port. The remote naming service (RNS), or middleware tier of the RPOS platform, essentially serves as a communication bridge between the clients and the application server.

Service implementations

Service implementations

The following diagram describes an implementation of services.



Services implementation

Encryption strategy

This version (as reflected on the title page) of the RPOS software is capable of the following:

1 Operating on a self check out solution.

2 Providing the ability to encrypt based on the algorithms supported by the Java Cryptography Extension (JCE).

Technical support services

Technical services hold the application together by providing common services to the application, services that are not necessarily driven by business requirements. In order to increase the maintainability of the code, a number of base technical services are provided in RPOS. They are described below.

Offline capabilities

In the event of a network or system failure, RPOS's Java-based multi-tiered platform allows retailers to run a register offline. An RPOS client automatically detects a network outage or failure in the backend systems and attempts to connect to other available application services. Further failure results in the client's switching to the customer defined downtime functionality, and transactions begin queuing locally.

RPOS captures transaction data from a client register and stores the data in a Java serialized object format. All transactions performed while a register is in offline mode are stored locally. These objects are either processed in real-time or written to local storage in an XML or sequential flat file format.

The mission critical design of RPOS guarantees that all data is delivered to the back-end repository or stored locally and queued until the network or back-end repository is available. When the client register automatically detects that the network or back-end repository is available, all locally stored transactions are automatically posted in sequential order to the back-end without any user intervention or management.

The more offline functionality desired, the more data (potentially) must be stored on the client. For example, a customer's requirements might state that a register should be able to continue to ring sales in the event of a network outage. This required the price look up (PLU) table and a subset of the company's employee table (for operator and salesperson lookups) to reside on the client.

When normal connections are restored, RPOS recognizes the online state and uploads all stored transactions for processing.

Logging service

This service provides the system with a standard method for logging information to a flat text file.

Internationalization service

This service uses Java classes with a .java extension to provide configurability for on-screen messages (such as on screen labels or error messages). To change the language for the RPOS GUI screens, the retailer can edit these .java files and recompile them without impacting the business functionality of the application. Note that although this service supports any number of languages, the screen flow remains left to right, top to bottom. RPOS uses standard Java resource bundle support.

Security service

The security service provides basic authorization and authentication functionality during user logon. The association of the user to security roles controls user access to the functional areas of the application. The security service validates a user's identity against a security store and retrieves the role memberships and role authorizations for that user upon a successful logon.

RPOS-related Java terms and standards

RPOS is deployed using the technologies, methods, versions and/or design patterns defined in this section.

Data access object (DAO)

This design pattern isolates data access and persistence logic. The rest of the component can thus ignore the persistence details (the database type or version, for example).

Java Development Kit (JDK), version 1.4.1

Standard Java development tools from Sun Microsystems.

Java Messaging Service (JMS) topic

A JMS topic is message-oriented middleware. The topic can be thought of as broadcasting a message to RPOS clients.

JDBC

JDBC is a means for Java-architected applications such as RPOS to execute SQL statements against an SQL-compliant database, such as Oracle. Part of Sun's J2EE specification, most database vendors implement this specification.

Naming conventions in Java

- Packages: The prefix of a unique package name is written in all-lowercase letters.
- Classes: These descriptive names are unabbreviated nouns that have both lower and upper case letters.
- Interfaces: These descriptive names are unabbreviated nouns that have both lower and upper case letters. The first letter of each internal word is capitalized.
- Methods: Methods begin with a lowercased verb. The first letter of each internal word is capitalized.

Persistence

The protocol for transferring the state of a business object between variables and an underlying database.

Persistent connections

The state of connection between an application and the database.

Remote interface

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

Remote Method Invocation (RMI)

RMI offers a simple and direct model for distributed computation with Java objects. Remote method invocation (RMI) is the action of invoking a method of a remote interface on a remote object. Most importantly, a method invocation on a remote object has the same syntax as a method invocation on a local object.

Skeleton

The skeleton understands how to communicate with the stub across the RMI link. The skeleton performs all of the following:

- 'Talks' with the stub.
- Reads the parameters for the method call from the link.
- Makes the call to the remote service implementation object, accepts the return value, and then writes the return value back to the stub.

Stub

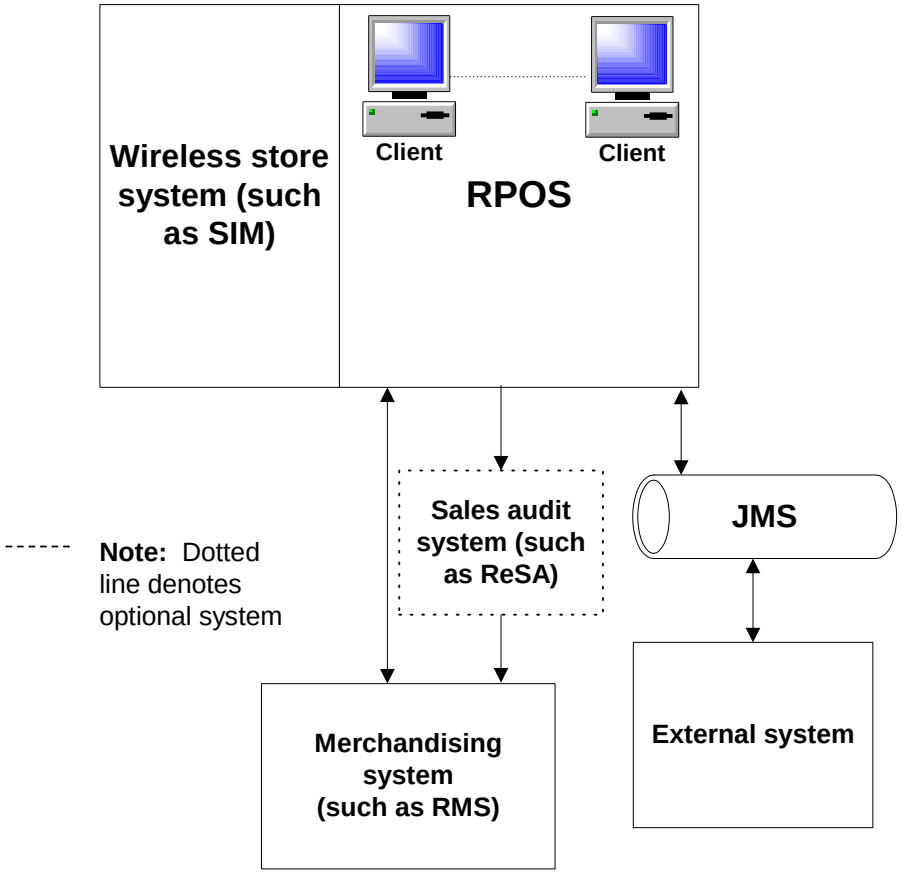
The stub contains information that allows it to connect to a remote object, which contains the implementation of the methods. The stub implements the same set of remote interfaces as the remote object's class.

Chapter 4 – Integration interface dataflows

Overview

This chapter provides an overview as to how RPOS is functionally integrated with other systems (including other Retek systems). The discussion primarily concerns the flow of RPOS-related business data across the enterprise.

A diagram shows the overall direction of the data among the products. The accompanying explanations of this diagram are written from a system-to-system perspective, illustrating the movement of data. Note that this discussion focuses on the functional use of data; the means of data movement (for example, batch) is not illustrated in this chapter.



RPOS-related dataflow

Note: Because RPOS and SIM can share a database, the dataflow between the systems can be substantial. The data that is described below is not comprehensive but presents a high-level view of some of the key exchanges between the systems and the distinctions each makes regarding the data.

From RPOS to a wireless store system (such as SIM)

- Stock-on-hand (SOH) data
  Stock-on-hand updates, due to sales and returns, are reflected by SIM within its GUI. Stock-on-hand data is held in quantities within buckets (for example, available, unavailable, damaged, reserved, and so on).
- New employee data
  Employees created in RPOS can receive logon privileges in SIM. Note though that this dataflow does not apply if the SIM system is integrated with LDAP.

From a wireless store system (such as SIM) to RPOS

- Stock-on-hand (SOH) data
  Stock-on-hand updates, due to transfers and so on, are communicated to RPOS.
- Item data
  - Description of the item
  - Item code
- Location data
  To RPOS, location data consists of updated store information. RPOS does *not* recognize warehouse data.
- Item-location association and price data
  For the purposes of RPOS, this data is related to specific items in a store. For example, in this Canadian store, an item is being sold for this many Canadian dollars.
- Tax data at the location level
  For RPOS's purposes, a given store has a tax rate of this amount.
- Promotions data and promotions-item association data
  RPM both defines promotions and links promotions to items. RPOS applies this data in transactions.

From to RPOS to a merchandising system (such as RMS) or to an (optional) sales audit system (such as ReSA)

- Transaction data
  RPOS exports a transaction log (in the RPOS upload format) with data such as all sale and return transactions that the system has processed. Note that complete layaways are treated as sales. Retek Sales Audit (ReSA) is a tool that monitors the reliability and accuracy of transaction data and compares the data to the rules and guidelines that a retailer establishes. ReSA flags inaccurate data for sales auditors, who can then correct the errors.

From RPOS client to the JMS queue

- Sender data
  Senders send data messages to topics and queues on a Java Messaging Service (JMS) server. For more information regarding messaging, see “Chapter 6 – Messaging framework”.
  - Logging-related data
    The information sender returns the contents of the client log files. Through mission control, the retailer can access this data for remote administration purposes. Examples of information include starting these services, going online, and so on.
  - Exception-related data
    The exception sender returns the contents of the client’s error log files. Through mission control, the retailer can access this data for remote administration purposes. Exceptions are created when problems occur with the system.

From the JMS queue to the RPOS client

- Receiver data
  Receivers receive data messages to topics and queues on a Java Messaging Service (JMS) server. For more information regarding messaging, see “Chapter 6 – Messaging framework”.
  - Corporate message data
    The retailer can send a text string to the front screen of the clients with any message that suits business needs, (a welcome message, the latest news, reminders, and so on).
  - Exception clearing
    This message clears the error log so that a client does not have to continue to see exceptions that have already been addressed and/or resolved.
  - Price update
    An external system can publish price updates to the JMS queue, and RPOS can subscribe to get price changes. RPOS uses the information to update its item file, which contains the price data.

From RPOS client to RPOS client

Peer-to-peer functionality within RPOS is both configurable and extensible. The dataflow below is what is configured as ‘out of the box’ functionality. For more information, see ‘Clientmaster configuration file (client_master.cfg)’ in “Chapter 2 – Backend system administration and configuration”.

- Suspend/recall data
  When a user is in the middle of a transaction on one client and suspends it, the user can go to another client and recall the same transaction.
- Updated employee data
  If a client does not have updated employee data, a client can access the data from another client to find the employee data.
- Manifest update
  The Manifest can contain any combination of files, including the following: software updates, images, binary files, text files, Java Virtual Machines (JVMs), PLU updates, and so on.

Chapter 5 – Functional overviews

This chapter provides information concerning the various aspects of RPOS functionality.

RPOS management

RPOS management	
Transaction management	Performs financial tasks.
Layaway management	Prints layaway reports and returns a layaway item to stock.
Employee management	Manages your employees and access and complete various employment forms.
Timecard management	Manages your employees and timecards.
Store goals	Manages sales goals for the store and employees.
View receipt log	Allows for the viewing of receipts printed store-wide.
End of session	Closes a register.
End of day	Records end-of-day totals and deposits.
Reports	Allows you to monitor and track a store's performance, consumer trends, and so on.

Transaction management

In addition to performing transaction functions, RPOS also allows a manager to perform financial tasks specific to his or her managerial duties. Among these tasks are the following:

- Voiding transactions
- Recording paidouts, which are given in the form of cash, for store expenses
- Recording cash drops, or nightly deposits, into the system at the end of the business day
- Collect incoming money not resulting from transactions

Layaway management

The layaway management features in RPOS enable you to perform the following tasks:

- Return a layaway item to stock
- Print reports for outstanding layaway payments
- Print reports for overdue layaway payments

In instances where a customer has put an item on layaway and decided not to purchase that item, RPOS will have the capability to return that item to stock. Also, the layaway reporting feature in RPOS helps manage the outstanding and overdue layaway transactions.

Employee management and security

The employee management option allows you to manage your employees and to access and complete various employment forms including:

- Employee applications: This feature enables you to view current employee data or to enter new employee data into the system.
- Time off request: This feature enables you to view existing time off requests or submit new time off requests (vacation, sick leave, personal leave) into the system.
- This feature allows you to view and modify an employee's W-4 data.
- This feature allows you to view and modify an employee's I-9data.

Employee security is also maintained in RPOS. Employee security determines which parts of RPOS various employees can access. Within the employee security portion of RPOS you are able to view current employees' security status, create new employees, assign and modify security roles for employees create or terminate employee status.

Employee schedules

RPOS's labor scheduling features optimize employee resources for labor management and provide you with the ability to plan, track, and analyze labor resources and scheduling. Using RPOS, labor scheduling is a two-fold process. The employee scheduling information option is used in conjunction with the maintain schedule option to create, maintain, generate, and print schedules. By accessing and editing the following information, you are able to add, modify, and maintain employee schedule resources for each employee.

Employee schedule resources	
Employee name	Store number and name
Store phone number	Target and maximum hours
Current employee access role or roles assigned to the employee	Current employee role or roles assigned to the employee
Current time-off requests assigned to the employee	Current re-occurring availability (preferred availability and unavailability) information

As schedule information is gathered for individual employees, RPOS also has the capability to generate schedules based on the user defined labor resource needs. Among the scheduling options in RPOS are the following:

Maintain employee schedules	
Create new schedule	Work with existing schedule
Copy existing schedule	Generate schedule
Manage schedule shifts	Modify schedule shifts
Delete a schedule shift from schedule	Manually assign employee schedule to shift
Unassign employee from schedule shift	Resolve schedule conflicts

Timecard management

This feature enables you to view employee timecards, modify employee timecards, view and modify employee benefit hours (sick time, personal time, and so on) and view timecard adjustments.

Store goals

The creation and maintenance of store and employee goals is also part of RPOS functionality. You can choose to view existing store goals, enter new store goals, modify employee goals, delete employee goals, restored deleted goals, and view the goaling graph.

View receipt log

This feature will generate a log for all receipts printed storewide. The receipts in the log are displayed as they would if they were printed.

Reports

RPOS allows you to view and print various reports so you can monitor and track activities such as store performance and customer buying habits. The following reports can be generated in RPOS:

- **Item code net sales**: Allows you to view all transactions for a specific item code.
- **Consultant net sales**: View all employees' yearly, monthly, weekly, and current net sales.
- **Net sales by transaction type:** Allows you to view a report of transaction types and the transaction type totals.

Customer management

This feature allows you to manage customer information. You can also view a customer's transaction history and merge customer information. The following table lists the various customer management options in RPOS.

Customer management	
Customer lookup	Allows you to search for customer information by either the customer’s phone number or the customer’s name.
Credit application	Submits an application for a store credit card for a customer.
Change quantity	Changes the quantity of a line item for a special order.
Delete	Deletes a line item from a special order.
Modify customer	Changes a customer's personal information. The customer's main phone number cannot be modified.
Modify info	Modifies the information for a special order.

RPOS process payments

In RPOS, there are numerous payment types, and a customer can pay for a transaction using a single payment type or a combination of payment types. Among these payment types are the following:

POS payment tenders	
Major credit cards	Stored value card
Debit cards	Traveler’s checks
Gift certificate/Gift card	Unlimited split tendering
Money order	Canadian cash/check
Electronic signature capture	Foreign tender support
Cash	Coupons/Trade certificates
In-house credit card	Store credit issue
Checks	User defined tender options

Note: Depending on the type of payment being made, there will be certain rules and restrictions for each of the different payment types. For example, if a customer is paying with a coupon, the coupon value cannot be greater than the amount due. In many instances, the RPOS system will alert you to these rules and restrictions.

RPOS start-and-end-of-day

When the store terminals are powered on at the start of a business day, a manager must perform the start-of-day functions to allow other users access to RPOS. Included in the start-of-day procedures are entering the store ID and password, the operator ID and the drawer fund amount. Many of these features are customizable and aspects such as the start/end drawer fund, employee security privileges, system updates, and register/operator/store level accountability can be modified by person(s) with those privileges.

Similarly, RPOS also performs many end-of-day tasks. Before end-of-day totals and deposits can be made, you are required to perform an 'end-of-session' for all but one of the registers. After this has been executed, the end of day reporting feature can be accessed in the management menu. Among the reports included are the over/short report, the bank report, and the currency media report. These reports can be generated after the deposit totals are displayed in RPOS and verified.

RPOS transactions

When a transaction is being processed after an item has been scanned, you have the capability to apply several different options to that transaction. After the transaction exists, you can view that transaction using various search options.

Transactions options

The transaction options allow you to apply different options to a transaction after you have scanned or entered an item code. The following table contains a list of the transaction options in RPOS:

Transaction options in RPOS		
Modify the quantity of purchase items	Apply discounts and markdowns	Delete a purchase item
Return a purchase item	Process a layaway transaction	Suspend a transaction
Assign a sales associate to a transaction	Assign a customer to a transaction	View line item details of a transaction
Cancel a transaction	Add miscellaneous items to a transaction	Add shipping information to a transaction

View transaction

You can view past transactions using the various search options below.

View transactions	
Discount-search for transactions with a specific discount type	**Payment type** - view transactions by payment type
Shipping- search for transactions with shipping information	**Transaction** – view transactions with a specific transaction option
Operator- view transactions by the operator who performed transaction	**Customer** – view transactions by customer
Consultant – view transactions by sales associate that performed transaction	**All** – view all transactions for a specific date

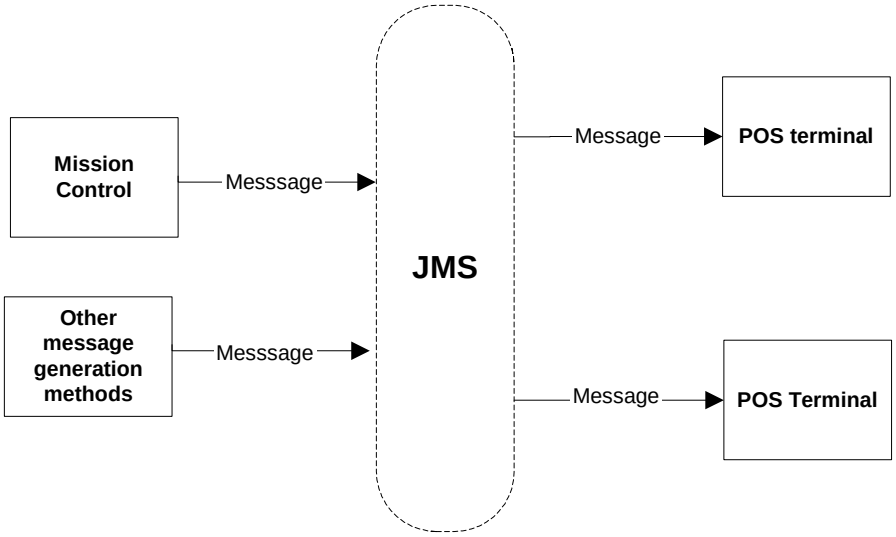
Merchandise return and even exchange

When merchandise is returned by a customer, the merchandise return functionality can be used. Functionality extends to the return and exchange of multiple items. Merchandise returns can be made with or without an exchange.

Chapter 6 – Messaging framework

Overview

RPOS provides the ability for registers and servers to send messages to topics and queues on a Java Messaging Service (JMS) server. As the diagram below illustrates, the messaging framework allows the clients to receive messages from topics and queues on a JMS server. RPOS uses messaging via JMS to perform tasks such as updating the client. For example, a price update during the day could be pushed out through messaging functionality. Messages can be created in Mission Control through the interface.



Messaging scenario in RPOS

Messaging functionality can be extended. JMS is capable of sending almost any object as a message's contents, though messages must meet specifications. For the purposes of simplification, RPOS has been designed to easily send messages in the form of java.lang.String and java.util.Properties objects.

JMS provides two types or models of messaging: point-to-point and publish/subscribe. In the point-to-point model, a message consumer establishes a named queue and listens for messages placed into the queue. In the publish/subscribe model, messages are published to and retrieved from named topics. RPOS uses the publish/subscribe model more frequently and thus provides more features for that model than the point-to-point model.

Message grouping

RPOS includes message grouping as an additional feature based upon the source or destination for a message. This feature is primarily used in the publish/subscribe model. A group is a named path similar to a Java package. The overall grouping structure is fairly arbitrary, but all groups must start at the same node (typically the company name). Each POS terminal has a group ID. A group specification for the Houston offices of the imaginary company Foo might look like the following:

```
Foo.USA.TX.Houston
```

In the RPOS system, a Store object knows about its group (see the method com.retek.iso.cr.store.Store.getGroupIdentifier()). Store clients are automatically grouped in their stores. A group specification for a client (number 101) in a store (number 1101) with the group identifier above would be:

```
Foo.USA.TX.Houston.Store.1101.TYPE_POS: 101
```

Message grouping is used by the messaging architecture in RPOS to allow message senders to target their messages to specific groups. For example, a message sent to the group `Foo.USA.TX.Houston` will reach all devices in the Houston group, including the example device above. However, this message would not be noticed at all by devices in the `Foo.USA.TX.Austin group.JMS.`

Publish/subscribe managed messaging

Managed messaging takes its name from the message manager which is the main base of operations for the sending and receiving of messages in this approach. It manages the message sender objects and the message receiver objects (collectively called ‘messengers’) that know what messages to send and what to do with incoming messages.

There are two message manager implementations: MessageManager and com.retek.iso.cr.node.ServerMessageManager.

The first implementation is used by the POS and reads its configuration information from the client_master.cfg configuration file. (For more information about the configuration of this file, see “Chapter 2 – Backend system administration and configuration”.) It reads the MESSAGE_RECEIVERS and MESSAGE_SENDERS properties in the file, and initializes the messengers from the classes listed in those properties. The second implementation is used by the application server container processes. The container reads its configuration from an XML file and puts the messengers elements as a list of classes into the com.retek.iso.cr.messaging.messengers.server system property. The message manager then reads this list and instantiate the messengers.

Preconfigured messengers

The RPOS system is preconfigured to send two types of messages. The first is an informational message. This message is sent to the VM_INFO topic. The second is a message on the status of the client, and this message type is sent to the EXCEPTION topic. Both the RPOS client and services containers send these messages.

The system's types of preconfigured receivers are shown below.

Note: The actual names of the preconfigured receivers are *not* what is shown. The names below are used for illustration purposes.

- Receiver 1
  This receiver type clears the exception or error log on either the client or server. It listens for messages from the CLEAR_EXCEPTIONS topic and expects a properties object containing the single key CLEAR. If the value for the key is true, the errors are cleared.
- Receiver 2
  This receiver type allows properties in configuration files to be changed. It listens to the CONFIGURATION_CHANGE topic and expects a properties object that specifies the name of the configuration file to be changed, the key to change, and the new value.
- Receiver 3
  This receiver type runs in service container processes. It tells the container to make some or all of the services it contains available or unavailable.
- Receiver 4
  This receiver type runs on RPOS clients. It allows new messages from corporate headquarters to be displayed on the RPOS terminal.
- Receiver 5
  This receiver type listens for price updates for RPOS.

To receive or send other types of messages, you can create implementations of the abstract classes MessageReceiver and MessageSender respectively.

Receiving Messages

Classes that want to process incoming messages should subclass `MessageReceiver` and implement its four abstract methods: `init()`, `getTopic()`, get`Group()`, and `hasMessages(IMessageSubscriber)`.

MessageReceiver methods to implement

`init()`

The `init()` method is used to perform any initialization your receiver deems necessary. One task that should be performed is to set the value of the `repeatInterval` attribute. The repeat interval indicates the number of minutes between message checks. A value less than one (1) will cause the receiver to check for messages only once.

`getTopic()`

The `getTopic()` method should return a String indicating the name of the topic of interest to the receiver. A receiver should always return the same topic.

`getGroup()`

The `getGroup()` method returns the group to which the receiver belongs. This can be retrieved using the `MessagingGroupService`. For example:

```
MessagingGroupService.getCurrent().getDevice();
```

`hasMessages(IMessageSubscriber)`

The `hasMessages(IMessageSubscriber)` is called when the receiver checks for messages and finds that some are present. The messages themselves can be retrieved using the `getMessages()` method of the `IMessageSubscriber` interface. The java.util.ArrayList returned by this method should be filled with all of the string and/or properties objects that were sent as messages to that topic and that were also sent to the receiver's group or one of its group's ancestors and that have not been previously retrieved.

The contents of messages sent to a topic are not specified by RPOS, but by the sender of the message. Typically, though not always, one of the keys in a Properties message should be 'DEVICE' and its value is the String representation of the group information for the sender of the message.

Building a receiver step-by-step

The following six steps are required to create a new message receiver.

1 Subclass `MessageReceiver`.

2 Implement the `init()` method, being sure to set the `repeatInterval` appropriately.

3 Implement the `getTopic()` method to return the name of the topic of interest.

4 Implement the `getGroup()` method to indicate the group of the client.

5 Implement `hasMessages(IMessageSubscriber)` to extract the messages and take appropriate action.

6 Add the fully-qualified class name of the new receiver to the MESSAGE_RECEIVERS key in client_master.cfg.

Sending messages

Classes that want to send messages should subclass MessageSender and implement its four abstract methods: `init()`, `getTopic()`, `getGroup()`, and `getMessage()`.

MessageSender methods to implement

`init()`

The `init()` method is used to perform any initialization your receiver deems necessary. One task that should be performed is to set the value of the `repeatInterval` attribute. The repeat interval determines the number of minutes the RPOS messaging architecture will wait between calls to the `getMessage()` method. A value less than one (1) will cause the `getMessage()` method to be called only once.

`getTopic()`

The `getTopic()` method should return a String indicating the name of the topic to which messages will be posted. A sender should always return the same topic.

`getGroup()`

The `getGroup()` method should return the group to which messages will be sent. Returning null will cause the message to be sent to all groups.

`getMessage()`

The `getMessage()` method is called by the RPOS messaging architecture to ask the sender if it has a message to send. Returning null indicates that no message is to be sent. Although the declared return type for this method is Object, currently, the RPOS messaging architecture only fully supports String and Properties objects. All other types are converted to String objects using the `toString()` method and are published to the topic in that form.

Building a sender step-by-step

The following six steps are required to create a new message sender.

1 Subclass `MessageSender`.

2 Implement the `init()` method, being sure to set the `repeatInterval` appropriately.

3 Implement the `getTopic()` method to return the name of the topic that will receive the messages.

4 Implement the `getGroup()` method to indicate the group that should receive the messages.

5 Implement `getMessage()` to return a String or Properties object to be used as message content.

6 Add the fully-qualified class name of the new receiver to the MESSAGE_SENDERS key in client_master.cfg.

Chapter 7 – Manifest deployment process

Software updates and data file updates are pushed out from the application server in an automated process called a Manifest. The Manifest can contain any combination of files, including the following: software updates, images, binary files, text files, Java Virtual Machines (JVMs), PLU updates, and so on. The process uses a combination of TCP/IP and UPD/IP for peer-to-peer (P2P) pushes of the Manifest between client terminals. Mission control can be used to create/generate manifests. The format of the manifest files is highly specific.

There are two types of Manifest updates: full and incremental. A Manifest is pushed out to the clients in a two-phase process. Each client is scheduled to check for updates at a pre-scheduled time determined by a pluggable daemon thread.

If there is more than one client in a store, the first client that receives the Manifest locks out all other clients in that location from receiving the Manifest. Once the first client receives the Manifest, it uses P2P networking to update the remaining in-store clients. The initial push sends the updates in a ‘wait-to-install’ state that allows system administrators time to check the Manifest logs and verify that all clients receive the updates and that there are no network or hardware outages.

At the system administrators’ discretion, the Manifest status is either changed to install on the application server, or the update is completely removed. When each client checks in at its pre-scheduled time and sees the status as ‘install,’ it installs the new software or data files. The system administrator can now view the Manifest logs to determine if there are any errors or clients that did not install the Manifest properly.

Note: Before you begin the Manifest process, you must ensure all RPOS files residing on each client terminal are writeable. If the RPOS files are read-only, RPOS will not be able to update itself and the Manifest will not work.

The two-step Manifest process

Create the Manifest

The first step of the Manifest is to create the Manifest that contains all the software updates and data file updates. The Manifest and the updated files will be saved on the application servers.

To create a Manifest:

1 On all application servers, create the following directory: /rpos/update/manifest

2 Create a .dat file called manifest.dat. The manifest.dat file will contain the parameters for each Manifest.

3 Create a directory for each Manifest if there are multiple Manifests. Each Manifest directory will contain the software updates and data file updates for that Manifest.

4 Within the manifest.dat file, create parameters for each Manifest. The parameters for each Manifest defines the parent directory for the Manifest, the Manifest status, and the Manifest date.

Note: When you create the Manifest, it is recommended that you set the value for the Manifest status to WAIT and that you set the value for the Manifest date to a date prior to the actual installation of the updated files on the client side. This helps ensure that all client terminals have successfully downloaded the Manifest and that the application servers have been updated with the new code.

Note: Once you change the status of a Manifest from WAIT to another status, we do not recommend that you change the status back to WAIT. If you do so, there is a possibility that the clients will become out of sync and run different versions of RPOS.

Push the Manifest to the clients

To push the Manifest to the clients:

1 Install the Manifest patch on all the clients.
All clients must be updated with an applicable patch .jar before they can download a Manifest. You will only have to install the Manifest patch once. After you install the Manifest patch, the clients are ready to download future Manifests. The following classes are needed so the client will download and restart the Manifest properly:

- BrowserManager.class
- IBrowserManager.class
- UpdateBootstrap.class
- UpdatePeerRmiServerImpl.class
- UpdatePeerRmiServerImpl_Skel.class
- UpdatePeerRmiServerImpl_Stub.class
- UpdateRMIServerImpl_Skel.class
- UpdateRMIServerImpl_Stub.class
- ClientManage.class

2 Install the Manifest patch on all the application servers.
The class files and configuration file are needed to be able to create a new UpdateService component to be added to a container. The clients will bind to this service during the UpdateBootstrap and UpdateDaemon processes.

3 Update the application servers.
After all the clients have installed the Manifest patch and are ready to download the Manifest, you can update the application servers with the updated code. A few days before you want to upgrade to the new version, copy the rpos/update and its content onto the application servers which are still running the old version of RPOS. The file rpos/update/manifest/manifest.dat should be updated. The clients will begin downloading the Manifest a few hours after the end-of-day is done (or during the start-of-day process). All files will be downloaded and stored in directory rpos\download. On the night you want to update to the new version of RPOS, make sure the clients have performed the end-of-day process. The next day, the clients will begin the installation process when start of day begins.

Chapter 8 – Java batch processes

This chapter provides the following:

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

Batch processing overview

RPOS's batch processes are run in Java. For the most part, batch processes engage in their own primary processing. However, there are some calls from the batch processes which utilize code from the normal services that are running in the server. Usually, this processing occurs when the batch processes engage in actions outside their primary processing (for example, when they utilize a helper method, touch the database, and so on).

Note the following characteristics of the RPOS's batch processes:

- They are not accessible through a graphical user interface (GUI).
- They are scheduled by the retailer.
- They are designed to process large volumes of data, depending upon the circumstances and process.

Running a Java-based batch process

For Unix systems, Java processes are scheduled through executable shell scripts (.sh files). For Windows systems, Java processes are scheduled through executable batch files (.bat files).

Retek provides the shell scripts (.sh files) and batch files (.bat files). They perform the following internally:

- Set up the Java runtime environment before the Java process is run.
- Trigger the Java batch process.
- Those processes that are 'download' processes bring data into RPOS from an external system such as a merchandising system (RMS, for example). Those that are 'uploaded' export data to an external system such as a merchandising system.

Command line parameter notes

Note the following information regarding command line parameters and the batch processes that are described in this chapter:

- For UpdateStoreDataFiles and the PosUpldGenerator, no parameter is necessary. By default, they process for all locations. However, you can specify one or more store numbers (IDs such as 5000, for example) as a command line parameter.

Summary of executable files associated to Java packages and classes

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

Executable shell script	Executable batch file for windows	Java package	Class
UpdateStoreDataFiles.sh **Note**: This command (and the equivalent .bat command) runs four batch processes, which are shown in the class column of this table.	UpdateStoreDataFiles.bat	com.retek.iso.cs.cs	• UpdateStoreEmployeeFiles • UpdateStoreItemFiles • UpdateStorePromotionFiles • UpdateStoreThreshholdPromotionFiles
PosUpldGenerator.sh	PosUpldGenerator.bat	com.retek.iso.cs.rms	PosUpldGenerator

Scheduler and the command line

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

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

Usage instructions are provided at the command line if incorrect parameters are passed.

Return value batch standards

The following guidelines describe the function return values and the program return values that RPOS's batch processes utilize:

- 0 - The function completed without error, and processing should continue normally.
- 1 - A non-fatal error occurred (such as validation of an input record failed), and the calling function should either pass this error up another level or handle the exception.

Functional descriptions and dependencies

The following table summarizes RPOS's batch processes and includes both a description of each batch process's business functionality and its batch dependencies.

Batch process	Details	Batch dependencies
PosUpldGenerator	This batch process exports a transaction log with data such as all sale and return transactions that the system has processed. The batch process writes the data to flat files in POS upload format. This batch process produces files with names that include a time-date stamp and the store number. See "Appendix A – POS upload file layout specification" later in this document.	Ad hoc
UpdateStoreDataFiles	This batch process generates XML files that are used by the RPOS clients (such as cash registers) in each store. The retailer can specify what stores the files are to be used in. Files that are generated include the following: • Items • Employee • Promotions • Threshhold promotions	Daily

A note about multi-threading and multiple processes

RPOS's batch processes are currently *not* set up to be multi-threaded or to undergo multi-processing.

A note about restart and recovery

RPOS Java-based batch processes do not utilize any type of restart and recovery. Rather, if a restart is required, a batch process can simply be restarted.

Batch logging

Log files are located in the same directory as the file being processed.

The system logs the following two types of messages for some batch processes:

- Warnings
  Warnings are *not* as serious as errors. They are logged for informational purposes, and they do not affect the running of the batch process.
- Errors
  Errors are the more serious of the two types of error messages. If these occur, they *do* affect the processing of the file.

Appendix A – POS upload file layout specification

Flat file used in the PosUpldGenerator batch process

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	File Type Definition	Char(4)	POSU	Identifies file as 'POS Upload'
	File Create Date	Char(14)	create date	Date file was written by external system
	Location Number	Number(10)	specified by external system	Store identifier
	Vat include indicator	Char(1)		Determines whether or not the store stores values including vat. Not required but populated by Retek sales audit
	Vat region	Number(4)		Vat region the given location is in. Not required but populated by Retek sales audit
	Currency code	Char(3)		Currency of the given location. Not required but populated by Retek sales audit
	Currency retail decimals	Number(1)		Number of decimals supported by given currency for retails. Not required but populated by Retek sales audit

Record Name	Field Name	Field Type	Default Value	Description
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies transaction record type
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	Transaction Date	Char(14)	transaction date	Date sale/return transaction was processed at the POS
	Item Type	Char(3)	REF ITM	Item type will be represented as a REF or ITM
	Item Value	Char(25)	item identifier	The id number of an ITM or REF
	Dept	Number(4)	Item's dept	Dept of item sold or returned. Not required but populated by Retek sales audit
	Class	Number(4)	Item's class	Class of item sold or returned. Not required but populated by Retek sales audit
	Subclass	Number(4)	Item's subclass	Subclass of item sold or returned. Not required but populated by Retek sales audit
	Pack Indicator	Char(1)	Item's pack indicator	Pack indicator of item sold or returned. Not required but populated by Retek sales audit
	Item level	Number(1)	Item's item level	Item level of item sold or returned. Not required but populated by Retek sales audit

Record Name	Field Name	Field Type	Default Value	Description
	Tran level	Number(1)	Item's tran level	Tran level of item sold or returned. Not required but populated by Retek sales audit
	Wastage Type	Char(6)	Item’s wastage type	Wastage type of item sold or returned. Not required but populated by Retek sales audit
	Wastage Percent	Number(12)	Item’s wastage percent	Wastage percent of item sold or returned. Not required but populated by Retek sales audit
	Transaction Type	Char(1)	‘S’ – sales ‘R’ - return	Transaction type code to specify whether transaction is a sale or a return
	Drop Shipment Indicator	Char(1)	'Y' 'N'	Indicates whether the transaction is a drop shipment or not. If it is a drop shipment, indicator will be 'Y'. This field is not required, but will be defaulted to 'N' if blank.
	Total Sales Quantity	Number(12)		Number of units sold at a particular location with 4 implied decimal places.
	Selling UOM	Char(4)		UOM at which this item was sold.
	Sales Sign	Char(1)	‘P’ - positive ‘N’ - negative	Determines if the Total Sales Quantity and Total Sales Value are positive or negative.

Record Name	Field Name	Field Type	Default Value	Description
	Total Sales Value	Number(20)		Sales value, net sales value of goods sold/returned with 4 implied decimal places.
	Last Modified Date	Char(14)		For VBO future use
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies transaction record type
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	Promotional Tran Type	Char(6)	promotion type – valid values see code_detail table.	Code for promotional type from code_detail, code_type = 'PRMT'
	Promotion Number	Number(10)	promotion number	Promotion number from the RMS
	Sales Quantity	Number(12)		Number of units sold in this prom type with 4 implied decimal places.
	Sales Value	Number(20)		Value of units sold in this prom type with 4 implied decimal places.
	Discount Value	Number(20)		Value of discount given in this prom type with 4 implied decimal places.
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	Transaction Count	Number(6)	specified by external system	Number of TDETL records in this transaction set
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type

Record Name	Field Name	Field Type	Default Value	Description
	File Line Identifier	Number(10)	specified by external system	ID of current line being processed by input file.
	File Record Counter	Number(10)		Number of records/transactions processed in current file (only records between head & tail)