

Retek[®] Invoice Matching 10.2



Operations Guide



The software described in this documentation is furnished under a license agreement, is the confidential information of Retek Inc., and may be used only in accordance with the terms of the agreement.

No part of this documentation may be reproduced or transmitted in any form or by any means without the express written permission of Retek Inc., Retek on the Mall, 950 Nicollet Mall, Minneapolis, MN 55403, and the copyright notice may not be removed without the consent of Retek Inc.

Information in this documentation is subject to change without notice.

Retek provides product documentation in a read-only-format to ensure content integrity. Retek Customer Support cannot support documentation that has been changed without Retek authorization.

Corporate Headquarters:

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

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

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

Sales Enquiries:
+44 (0)20 7563 46 46
Fax: +44 (0)20 7563 46 10

Retek[®] Invoice Matching[™] is a trademark of Retek Inc.

Retek and the Retek logo are registered trademarks of Retek Inc.

This unpublished work is protected by confidentiality agreement, and by trade secret, copyright, and other laws. In the event of publication, the following notice shall apply:

©2003 Retek Inc. All rights reserved.

All other product names mentioned are trademarks or registered trademarks of their respective owners and should be treated as such.

Printed in the United States of America.



Customer Support

Customer Support hours:

Customer Support is available 7x24x365 via e-mail, phone, and Web access.

Depending on the Support option chosen by a particular client (Standard, Plus, or Premium), the times that certain services are delivered may be restricted. Severity 1 (Critical) issues are addressed on a 7x24 basis and receive continuous attention until resolved, for all clients on active maintenance.

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

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

E-mail	support@rettek.com
---------------	--------------------

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

Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403
-------------	---

When contacting Customer Support, please provide:

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

Contents

Chapter 1 – Introduction.....	1
Who this guide is written for.....	1
What is invoice matching?	2
A note about Retek-based enterprises	2
Technical architecture overview	3
Presentation layer	3
Middle tier	4
Data access layer (DAL).....	4
Database layer	4
Technical services	4
Functional integration and dataflow.....	4
Where you can find more information	5
Chapter 2 – Initial implementation and system parameters 7	
System assumptions	7
reim.properties file	8
Connection information for the database.....	8
Authentication section	8
Minimum and maximum pool size.....	9
Security ports.....	9
Standard formats.....	9
Language to be loaded.....	10
Date formats for specific locales	10
Set the end of week day for the system	10
Batch log and error file paths	10
Error logging settings	11
Locking timeout variable.....	11
Auto-match threading options	12
Generic threading options (EdiUpload and AutoMatch).....	13
Cache sizes for translation service.....	13
system.properties file	13
Determine which general ledger (GL) options are dynamic.....	14
Data translation values.....	15
Child invoice indicator	15
Set the audit period.....	15
Mapping of document types to action codes	15

Chapter 3 – Technical architecture.....	17
Overview	17
The layering model.....	18
Presentation layer	19
Middle tier	19
Data access layer (DAL).....	20
Database layer	20
Technical services	21
Third party libraries.....	23
ReIM-related Java terms and standards	24
Chapter 4 – Functional design	27
Dataflow overview	27
From the supplier (to EDI) to ReIM	28
From ReIM (to EDI) to the supplier	28
From ReIM to the financial (AP/GL) staging table.....	28
From the client to ReIM	29
From the merchandising system to ReIM.....	29
From ReIM to receiver unit and cost staging tables	31
From ReIM to the merchandising system.....	31
Invoice matching process flow.....	32
The auto-match process.....	35
Overview	35
Cost pre-matching	36
PO location summary group matching	36
One-to-one invoice matching	39
Eligibility for line level matching.....	42
Line-level matching.....	43
Recycling and overall flow.....	46
Partially matched receipts.....	47
Tolerance	48
History and metrics.....	49
Best terms calculations.....	49
Overview	49
Terms ranking.....	49
Supplier options.....	49
Terms date	50
Terms-related data in the merchandising system (such as RMS)	51
Assumptions and dependencies	52
Primary tables involved	52

Chapter 5 – Interfaces and file layouts..... 53

The EDI interface and layouts.....	53
Overview	53
The EDI reject table.....	53
EDI invoice upload file layout.....	55
EDI invoice download file layout.....	74
The merchandise system interface	81
A word about indexes and RMS 9.....	81
Interface dataflows	81
Porting	82
DAL beans.....	82
Interface DAL porting example.....	83
Abstract beans	85
Concrete implementations of abstract beans	85
Porting concrete beans.....	92
Summary of porting steps for custom merchandising systems	93
Staging tables.....	93
Required merchandising system information	94
The financial system interface.....	101
Foundation financial data	101
Financial transactions	102
Major tables	102
LDAP and other user interfaces	103
LDAP.....	103
ReIM user table	103

Chapter 6 – Technical design..... 105

Services	105
Administration and configuration-related services.....	105
Batch entry-related services.....	106
Discrepancy-related services	108
Disputed credit memo-related services.....	109
Document-related services	109
EDI-related services	111
Financial-related services	112
Foundation-related services.....	113
Internationalization-related services.....	113
Invoice-related services	114
Matching-related services.....	115
Merchandising system-managed data.....	119
Non-merchandise document-related services	122
Reason codes-related services	122
Receipt-related services.....	123
Supplier-related services.....	123

User roles-related services.....	124
Locking design summary	124
Locking and tables.....	125
Locking management	126
Currency design summary.....	127
Merchandising system (such as RMS) and ReIM assumptions.....	127
Currency conversion process for amount tolerances	127
Currency-related system validations	128
Java currency formatting	128
Chapter 7 – Batch processes	129
Batch architectural overview.....	129
EDI-related file-based batch processes.....	129
Internal batch processes.....	129
Internal batch processes that write to staging tables.....	130
Batch names and Java packages.....	130
Functional descriptions and dependencies	131
Features of the batch processes	135
Scheduler and the command line.....	135
Batch return values	135
Batch log and error file paths	135
A note about multi threading.....	135
A note about restart and recovery.....	136
Batch purge	136
Overview	136
Assumptions and scheduling notes.....	138
Major modules.....	138
Primary tables involved.....	138
EDI invoice upload.....	138
Overview	138
Assumptions and scheduling notes.....	139
Restart and recovery	139
Primary tables involved.....	139
Receiver adjustment	140
Overview	140
Assumptions and scheduling notes.....	141
Primary tables involved.....	141

Auto-match (AutoMatchService)	141
Overview	141
The four algorithms	142
Assumptions and scheduling notes	143
Post processing	143
High-level flow diagram	143
Primary tables involved	144
Reason code action rollup	144
Overview	144
Assumptions and scheduling notes	145
High-level flow diagram	145
Primary tables involved	146
Disputed credit memo action rollup	146
Overview	146
Assumptions and scheduling notes	146
Primary tables involved	147
Resolution posting action rollup	147
Overview	147
Assumptions and scheduling notes	148
Primary tables involved	148
EDI invoice download	149
Overview	149
Assumptions and scheduling notes	149
Primary tables involved	150
Restart and recovery	150

Chapter 1 – Introduction

Retek Invoice Matching 10.2 (ReIM) represents the first version of an entirely rearchitected invoice matching application. ReIM is designed as a standalone application, with logic built into it that can reference any merchandising system.

ReIM lies between a retailer's merchandising system and its accounting operations. ReIM accurately and efficiently automates the verification process of receipts against supplier invoices and confirms the cost and quantity tolerance for the supplier. When the difference between an invoice and a receipt is outside of tolerance, the system allows for the recognition of discrepancies. The system includes a mechanism that allows its users to assign one or more reasons explaining the difference. If the reasons chosen include charging back a supplier or sending it payment, the system allows for the creation of a debit memo or a credit memo, respectively.

Through its technical and functional design, ReIM addresses the most common challenges posed by invoice matching:

- Matching accurately
- Processing efficiently
- Integrating with other systems

Who this guide is written for

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

- Business analysts who are looking for information about processes and interfaces to validate the support for business scenarios within ReIM and other systems across the enterprise (within a merchandising system such as RMS, for example).
- System analysts and system operation personnel
 - who are looking for information about ReIM's processes internally or in relation to the systems across the enterprise.
 - who operate ReIM on a regular basis.
- Integrators and implementation staff who have the overall responsibility for implementing ReIM.

What is invoice matching?

‘Invoice matching’ describes a control procedure designed to ensure the retailer pays the negotiated cost for actual quantities received. Invoice verification or matching is a fundamental and critical control procedure for every retailer.

ReIM allows multiple purchase orders and/or receipt numbers to be associated with an invoice. Through summary and detail level matching, invoice line items can be matched by unit cost and by quantity. Because ReIM includes an auto-match process, the client can focus its operations on discrepancy management.

Complete referential integrity is maintained with the referenced purchase orders and receiving documents. Neither POs nor receipts are purged if there are outstanding invoices. Open receipts are closed automatically after a specified length of time.

ReIM’s supplier tolerance functionality is able to process both total invoice cost or quantity and line item cost or quantity. Tolerances can be set by amount range and/or percentage. They can also be established in the client’s favor, rather than in the supplier’s favor.

Through electronic data interchange (EDI), ReIM can receive both invoices and credit notes, and it can send the supplier a debit memo, credit note request, or credit memo.

ReIM places matched invoices on staging tables that represent a standard interface to Oracle financial applications. Clients can then move the data to their financial system, at their discretion.

Because of the automation of standard tasks, efficiencies are increased and costs are reduced in the retailer’s organization. The ReIM solution value proposition focuses on return of investment through:

- Productivity gains from automatic matching and exception-based manual effort
- Efficient processes that help ensure discount and payments are met
- Accurate processing that ensures control objectives are met and that inventory is valued at lowest possible cost, improving gross margin

A note about Retek-based enterprises

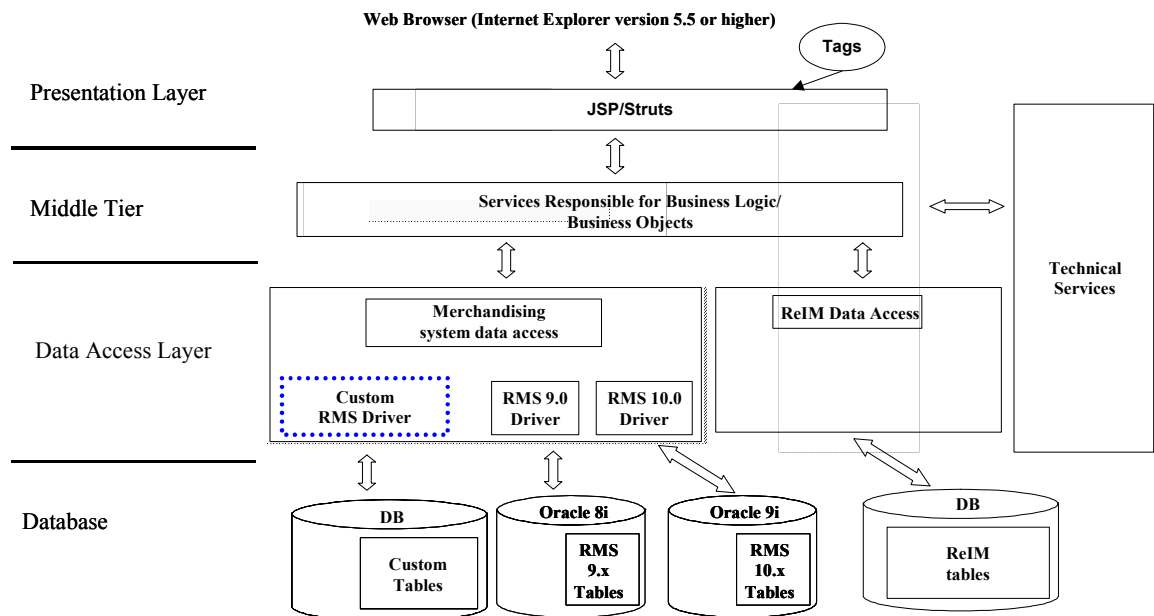
Although ReIM has been developed as a stand-alone product, the most efficient implementation would be as part of the Retek product suite. This integration provides the following important benefits:

- The number of interface points that need to be maintained is minimized.
- The amount of redundant data and processes within the retail organization is limited.
- Future enhancements allow for greater extensibility into the retail enterprise.
- Delays in product introductions can be minimized.

Technical architecture overview

ReIM utilizes a Java platform because it offers the optimum solution to the challenges presented by the need for database independence. The architecture is built upon a layering model. That is, layers of the application communicate with one another through an established hierarchy and are only able to communicate with neighboring layers.

The following diagram, together with the brief explanations that follow, offer a high-level conceptual view of the layers and their responsibilities within the architecture. For a more detailed description of this diagram, see Chapter 3, “Technical architecture”.



An overview of ReIM's layered architecture

Presentation layer

This area of the architecture encapsulates the graphical user interface (GUI) processing. A web browser accesses JSP pages using a Struts tag library. JSPs consist of JavaScript and standard HTML. Struts provides an open source framework for building Web applications. The presentation layer only interacts with the middle tier services.

Middle tier

The service layer consists of a collection of Java classes that implement business logic (data retrieval, updates, deletions, and so on) via one or more high-level methods. The service layer is the entry point to the middle tier and separates the presentation layer from the database layer.

Within ReIM, business objects are beans (that is, Java classes that have one or more attributes and corresponding set / get methods) that represent a functional entity. In other words, business objects can be thought of as data containers, which by themselves have almost no business functionality. The service layer may utilize more than one class from the data access layer in order to combine the data from more than one database table to fully populate a business object.

Data access layer (DAL)

Classes in the DAL abstract the actual persistence mechanism that is being used to persist business objects. The DAL provides the mechanism that allows ReIM to be associated to a different persistence engine.

The ReIM DAL consists of two very distinct portions: a DAL to ReIM ‘owned’ tables and an interface DAL to merchandising system tables.

Database layer

The database layer is the application’s storage platform, containing the physical data (user and system) used throughout the application. This layer is only intended to deal with the storage and retrieval of information and is not involved in the manipulation of the data.

Technical services

Technical services hold the application together by providing common services to the application, services that are not necessarily driven by business requirements. Technical services include application frameworks such as error logging, internationalization, transaction management, application security, and so on.

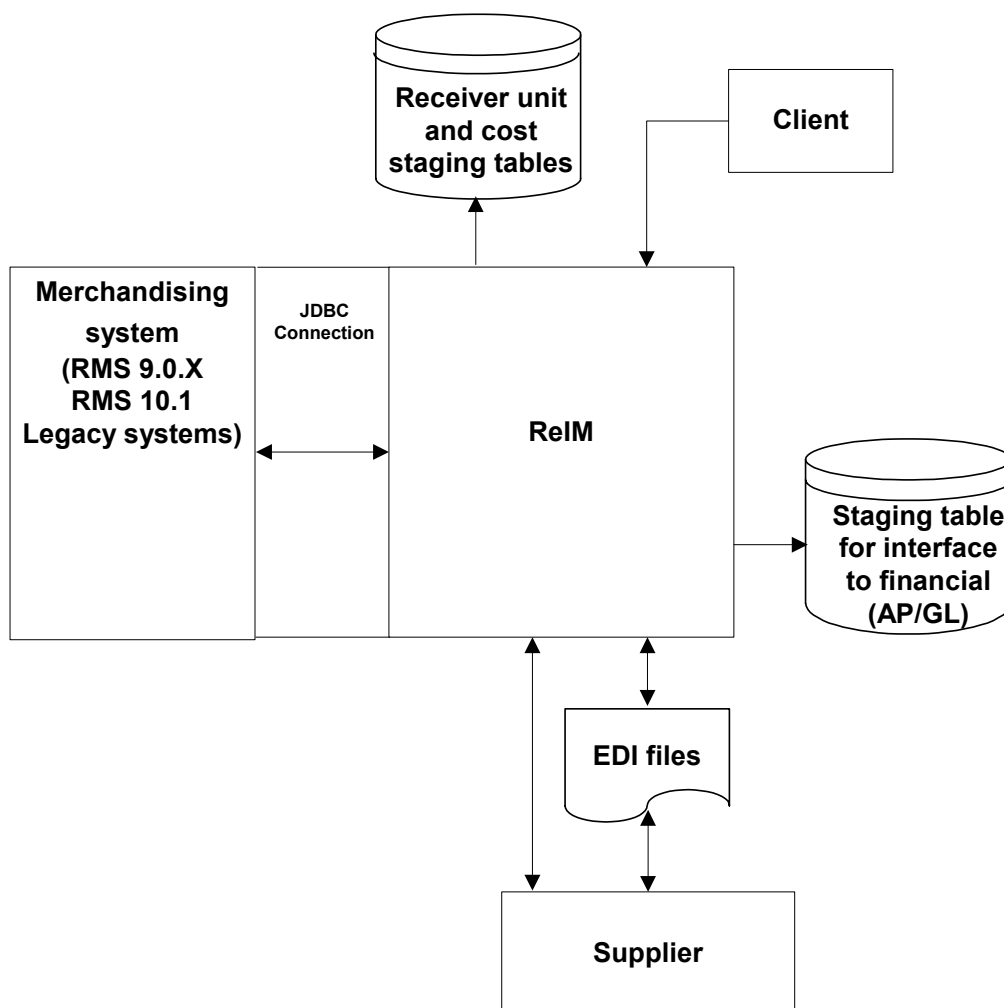
Functional integration and dataflow

The following diagram offers a high-level view of ReIM’s place in the enterprise. For a more detailed description of this diagram, see Chapter 4, “Functional design”.

For Retek enterprise clients, ReIM integrates with the Retek Merchandising System (RMS), which provides, among other data, foundation and purchase order (PO) data.

From a supplier, ReIM can receive external invoices via EDI or not via EDI (through paper, for example). Through ReIM processing or manual input, invoice data is inserted into the applicable ReIM tables.

ReIM places matched invoices on staging tables for use in the client's financial system at its discretion. The interface is standard for an Oracle financial application.



Where you can find more information

This guide does not show you how to use the front-end of ReIM. Rather, the focus here is on how data is managed, how it flows, and how it is processed. This guide does not explain, except at a high level, the ReIM-related data flow and processing that occurs among other applications across an enterprise (for example, within the merchandising system).

If you wish to find further information ReIM, see the following applicable Retek documents:

- ReIM front-end documentation (for example, the ReIM User Guide)
- ReIM Installation Guide
- Retek Merchandising System (RMS) product documentation

Chapter 2 – Initial implementation and system parameters

This chapter of the operations guide is intended for database administrators who provide database 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 that establish ReIM's environment.

System assumptions

- The invoice matching ind on the RMS system options table (SYSTEM_OPTIONS.INVC_MATCH_IND) should be set to No. Setting this value to Yes causes the limited invoice matching functionality in RMS to execute. Clients using ReIM should *not* allow the RMS-based invoice matching code to run.
- Currently, ReIM expects all invoices to be in eaches or the standard unit of measure (SUOM) converted to eaches. No other units of measure can be invoiced using ReIM.
- RMS buy/get deals are not supported in ReIM. Bill-back or rebate deals are also not supported. ReIM only supports off invoice deals.
- ReIM does not support invoices generated by ReSA for consignment goods. ReIM does not support any consignment functionality.
- ReIM uses non-merchandise codes defined on the RMS table NON_MERCH_CODE_HEAD. The form that allows users to enter non-merchandise codes in RMS is not available when the RMS invoice match ind is set to N (see above). Instead, non-merchandise codes should be added to the NON_MERCH_CODE_HEAD table using the database.
- A record must be inserted into the IM_SYSTEM_OPTIONS table in order for the application to log on successfully.
- GL options
If any segments are going to be dynamic, the client is responsible for maintaining the dynamic segment location and department tables.
- Supplier options
All suppliers must have options defined in order for their invoices to be processed by the system.
- GL account maintenance
All reason codes, non-merchandise codes, and basic transactions must be mapped through GL account maintenance for posting to succeed.
- Multiview
The Document Find, Group Entry List, and Group Entry pages allow the user to define how they want the screens to appear. The Multiview functionality allows the user to move fields around on the pages and save those views for future use. In order for Multiview to work and for these screens to populate correctly, IM_GLOBAL_PREFERENCES must be populated.

reim.properties file

Client-defined configurations for ReIM are located in the `reim.properties` file. The key system parameters contained in this file are described in this section.

Note that within the property file (and thus in some of the examples from that file below), a `#` sign that proceeds a value in the properties file signifies that what follows is a comment and is not being utilized as a setting.

Every setting in the `reim.properties` file is configurable. When clients build the code to their environment, they must update these values to their specific settings.

Connection information for the database

In this portion of the file, this data includes what JDBC driver the system is utilizing. (For more information about JDBC, see Chapter 3, “Technical Architecture”.) This data also includes what datasource (merchandising system) that ReIM is utilizing for its foundation data and the environment information associated to that datasource. The following settings apply:

- JDBC driver
- URL
- Datasource
- Username
- Password
- SchemaOwner
- BeanDriver

Authentication section

Authentication within ReIM addresses the legitimacy and the security privileges of users, an important aspect of the system’s security handling process.

Within the authentication section of this file, the client selects either LDAP or DATABASE, depending upon which is applicable.

For clients selecting LDAP, see Chapter 5, “Interfaces and file layouts” later in this document and see the ReIM Installation Guide for more settings and information.

Minimum and maximum pool size

The pool size pertains to the number of available database connections that the client intends to keep available in the pool. A system administrator is encouraged to adjust these values per configuration to match the client's anticipated number of users. The default values are intended to be a mere starting point.

In the example below, a minimum of 5 connections are available, and no more than 10 are available.

```
# Minimum and maximum pool size to maintain
pool.min=5
pool.max=10
```

Security ports

Within the enterprise, a port is an endpoint to a logical connection and the way a client program specifies a specific server program on a computer in the network. A security port is analogous to an address for a given machine. The security port 'listens' at this address, and if the system needs to process security-related data, it must 'talk' only to that address to do so. The security-related processing can only occur at the place where the 'listening' is occurring.

```
# Security Ports
security.ssl_mode=2
security.port_non_ssl=8080
security.port_ssl=8443
```

Standard formats

Batch date format

With regard to incoming EDI sent data, clients can define (through their vendors) the batch date format that they would prefer the system to receive. The system uses the batch date format that the operator enters in this section of the file. For example:

```
batch_date_format=yyyymmddhhmmss
```

Quantity decimals allowed

The database tables within the system allow for quantities to be held in decimals (for example, 12.5). Quantity decimals allowed means how many decimals the system displays for a quantity field. In general, quantity decimals are utilized by grocery clients.

This is an integer value. ReIM is set to 0 because the system assumes eights. If a client wished to show 4 decimals, the value would be 4.

Language to be loaded

This setting instructs the system as to where to go to get an applicable locale's properties file. This file includes the localized labels, select list boxes, and so on. For example, the system's default English file is `ReIMResources.properties`. Another example is `ReIMResources_fr_FR.properties` for French.

The applicable values follow Java standards for languages and countries. Languages are always two letters and lower case. Countries are always two letters and upper case.

For example:

```
# Which language should be loaded?
language=en
country=US
```

Date formats for specific locales

To provide a user-friendly localized date format that is understood by users, the client may select one of four date formats that are available. For any entry of dates through the GUI, the user can enter a date in the format defined. The applicable values include the following:

- `dd/MM/yyyy`
- `dd-MM-yyyy`
- `MM-dd-yyyy`
- `MM/dd/yyyy`

For example:

```
date_format=MM/dd/yyyy
```

Set the end of week day for the system

The system administrator establishes this value to inform the system what that end of the weekday is. Sunday is equal to 1, and Saturday is equal to 7.

Batch log and error file paths

Batch log file path

The name and directory of the batch log files are established through this setting.

For example:

```
batcherrorlogpath=/files0/ReIM10.2/dev/error
```

Batch error file path

The name and directory of the batch error files are established through this setting. All errors and all routine processing messages for a given program on a given day go into this error file.

For example:

```
batcherrorlogpath=/files0/ReIM10.2/dev/error
```

Error logging settings

Error logging threshold

The ReIMException class automatically logs itself to the application log file. The level of logging may be raised or lowered in the setting below. The operator's choice instructs the system to log that level of error *and errors above that level*.

In the example below, an operator has configured the system to only display ERROR messages and above:

```
ReIMLoggerLogLevel=ERROR
```

Because the logger reveals message priority levels using numbers, the word equivalents are shown below:

- UNKNOWN = -999
- FATAL = 2
- ERROR = 3
- WARN = 4
- VALIDATION = 5
- INFO = 6
- DEBUG = 7
- PERFORMANCE = 8

Locking timeout variable

When a user tries to commit information to the database, the system checks to determine that he or she continues to have a lock because locks can time out. The number of seconds until the time out occurs is the locking timeout variable, defined in this file. If the user no longer has a lock, the user receives a message saying that changes cannot be saved.

The client should set the timeout period that makes the most sense for its business needs. Ideally, the timeout period should be long enough so that users can finish working on one record, but short enough so that unintentional locks (during lunch, and so on) do not delay other users an inordinate amount.

During a session, when the system has been idle for longer than the locking timeout variable, the system does *not* release the lock. Rather, if a second user attempts to lock the same table, the system's locking service determines whether the locking timeout variable has been exceeded. If the locking timeout variable has been exceeded, the locking service continues locking the table but for the second user. For the first user, the lock on the table has expired.

ReIM has a lock table defined for the bulleted areas below. For example, the table IM_DOC_HEAD locks the corresponding values from IM_DOC_HEAD_LOCK. To establish the duration of the timeout, the client can enter a mathematical expression using the variables. In the list below, the client has set the tables, with one exception, to be released after one hour of session inactivity. The table is only released at the end of the user's session or, in the case of a system crash, at the beginning of the next user's session.

- business_roles_lock_timeout=1*hour
- doc_group_list_lock_timeout=1*hour
- doc_head_lock_timeout=no_expire
- edi_reject_doc_lock_timeout=1*hour
- supplier_options_lock_timeout=1*hour
- system_options_lock_timeout=1*hour
- tolerance_dept_lock_timeout=1*hour
- tolerance_supp_lock_timeout=1*hour
- tolerance_supp_trait_lock_timeout=1*hour
- tolerance_system_lock_timeout=1*hour

Locking timeout variables are in milliseconds. Conversion data is provided below for the client's convenience.

- millisecond=1
- second=1000
- hour=3600000
- day=86400000
- month=2592000000
- no_expire=-1

Auto-match threading options

These parameters are used to configure auto-match threading. Auto-match can either be run as a single thread, or it can be threaded by the location hierarchy. Currently, this parameter is defaulted to thread auto-match by district. Changing the thread parameter is as simple as commenting out one parameter and uncommenting another.

For example:

```
auto_match_thread_by=ThreadByDistrict
```

Generic threading options (EdiUpload and AutoMatch)

The `thread.backgroundThreadTimeout` parameter is currently only used by EdiUpload for rejection files. The value represents how long the log writing thread polls an empty work queue before shutting down. Units are expressed in milliseconds.

For example:

```
thread.backgroundThreadTimeout=1800000
```

The `thread.consumerThreadTimeout` parameter represents how long the consumer pool threads. The value is used for executing the transactions for both EdiUpload and AutoMatch. Units are expressed in milliseconds.

For example:

```
thread.consumerThreadTimeout=60000
```

The `thread.consumerThreadKeepAlive` parameter represents how long the consumer/worker stays alive. Units are expressed in milliseconds.

For example:

```
thread.consumerThreadKeepAlive=60000
```

The `thread.consumerThreadPoolMin` and `thread.consumerThreadPoolMax` parameters represent the range of consumer/worker threads that can be created for the pool.

For example:

```
thread.consumerThreadPoolMin=10
thread.consumerThreadPoolMax=100
```

Cache sizes for translation service

To enhance the system's performance speed, the system utilizes a cache when performing data translations into another language.

For example, suppose the system has been configured to offer French translations. When a French user encounters a location name, the system retrieves the translated location name from the database and then stores it in a cache. If the system needs to retrieve the same translated location name at a later time (for another user, for example), the system would retrieve it from the cache rather than from the database. This parameter represents the number of entries within the cache that the system is allowed to use for such processing.

For example:

```
translation.items_desc_cache_size=100000
```

system.properties file

This file includes system options settings that were *not* built into the graphical user interface (GUI) because they cannot be changed once ReIM has been implemented.

Determine which general ledger (GL) options are dynamic

The parameters in this section of the file determine whether the client's segments for the IM_GL_OPTIONS table are dynamic or non-dynamic.

If the client's segments are non-dynamic, all settings would equal 'N'.

If the client's segments are dynamic, note the following:

- The system allows a maximum of four segments that can be dynamic.
- Those values that are set to dynamic (that is, set to 'Y') can be associated to the following four concepts (note that company and location are always paired together and that department and class are always paired together):
 - Company
 - Location
 - Department
 - Class
- If the client's segments are dynamic by location, the location number is included in the parameter.

For example:

```
system.gl_option_dynamic_1=Y
system.gl_option_dynamic_2=Y
system.gl_option_dynamic_3=N
system.gl_option_dynamic_4=Y
system.gl_option_dynamic_5=Y
system.gl_option_dynamic_6=N
system.gl_option_dynamic_7=N
system.gl_option_dynamic_8=N
system.gl_option_dynamic_9=N
system.gl_option_dynamic_10=N
system.gl_option_dynamic_11=N
system.gl_option_dynamic_12=N
system.gl_option_dynamic_13=N
system.gl_option_dynamic_14=N
#Business concept mapping for dynamic segments
system.gl_option_dynamic_mapping_1=COMPANY
system.gl_option_dynamic_mapping_2=LOCATION
system.gl_option_dynamic_mapping_3=
system.gl_option_dynamic_mapping_4=DEPARTMENT
system.gl_option_dynamic_mapping_5=CLASS
system.gl_option_dynamic_mapping_6=
```



```

system.gl_option_dynamic_mapping_7=
system.gl_option_dynamic_mapping_8=
system.gl_option_dynamic_mapping_9=
system.gl_option_dynamic_mapping_10=
system.gl_option_dynamic_mapping_11=
system.gl_option_dynamic_mapping_12=
system.gl_option_dynamic_mapping_13=
system.gl_option_dynamic_mapping_14=

```

Data translation values

The client can establish whether the system should translate certain values in the system for the user into another language. The setting was designed because the translation of values can impact system performance. Note that the parameters in this section do *not* apply to the translation of the labels on the screens.

For example, if a non-English speaking user encounters an item list of values (LOV), the setting in this section determines whether the user could see a description of those items in the second language.

Note: The translated values must be initially set up in the merchandising system (such as RMS). The values shown are from the TL_SHADOW table in RMS.

Child invoice indicator

In this section, the client defines a string. When a parent invoice enters the system, the system can split the invoice into its child invoices. A parent invoice can contain many locations; a child invoice contains only one. The system continues to use the parent invoice ID, along with both the string that is defined by this parameter and the location number to which the child invoice is associated.

Set the audit period

The parameter determines how many days the system retains audit trail data.

For example:

```
system.purge_tolerance_audit_period=2
```

Mapping of document types to action codes

This is the default action based on document type which limits the reasons presented in the reason code list of values (LOV) on the document maintenance detail screen.

For example:

```

#CREDIT_NOTE_REQUEST_PRICE
CRDNRC=CBC

```


Chapter 3 – Technical architecture

This chapter describes the overall software architecture for Retek Invoice Matching. The chapter provides a high-level discussion of the general structure of the system, including the various layers of Java code.

Note that at the end of this chapter, a description of ReIM-related Java terms and standards is provided for your reference.

Overview

ReIM utilizes a Java platform because it offers the optimum solution to the challenges presented by the need for database independence. The architecture is built upon a layering model. That is, layers of the application communicate with one another through an established hierarchy and are only able to communicate with neighboring layers.

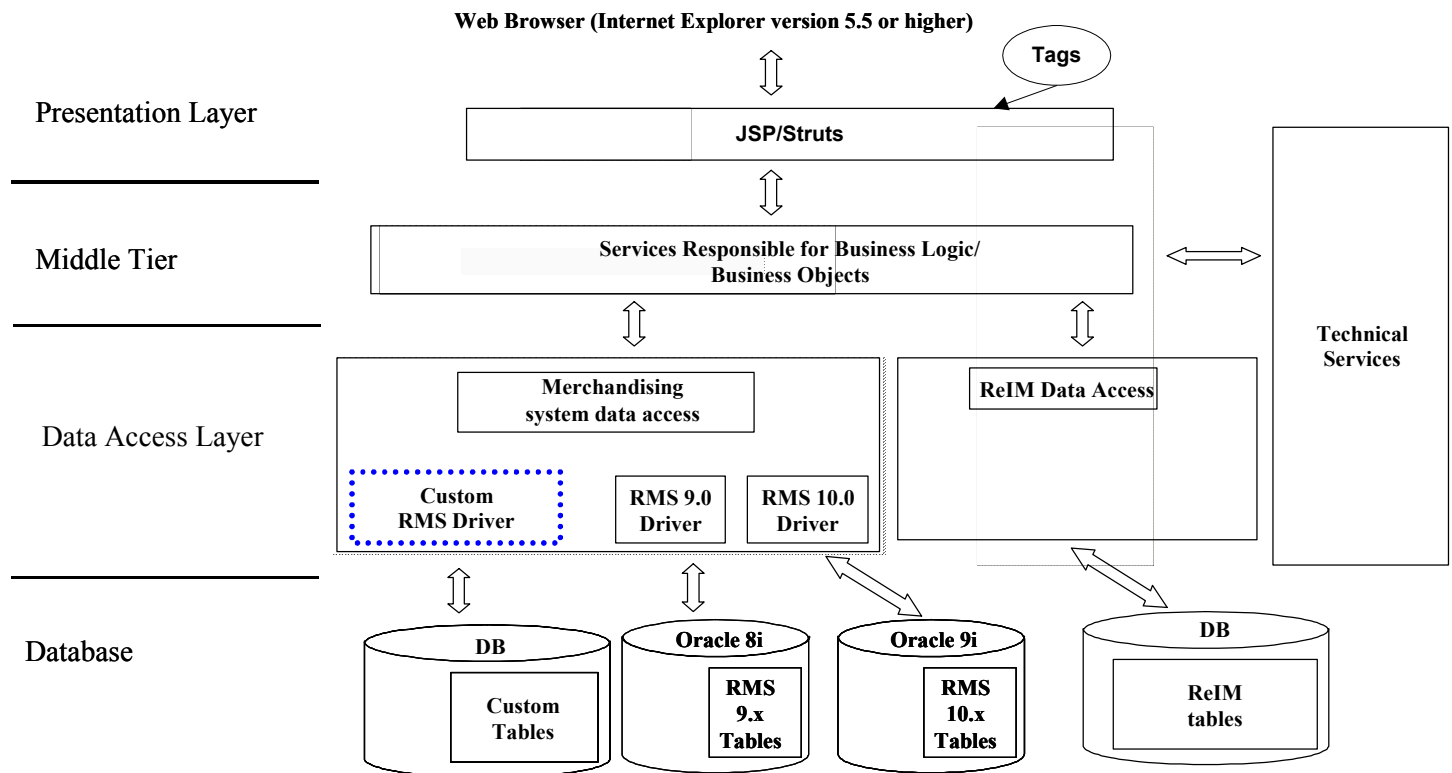
The application is divided into a presentation layer, a middle tier consisting of services and business objects, and a database access/driver layer. Technical services provide the ‘glue’ that holds the application together, offering the application frameworks for error logging, internationalization, transaction management, application security, and so on.

The segregation of layers has the following advantages, among others:

- The separation of presentation, business logic, and data makes the software cleaner, more maintainable, and easier to modify.
- The look and feel of the application can be updated more easily because the GUI is not tightly coupled to the back end.
- A layered architecture has become an industry standard.
- Portions of the data access layer (DAL) can be radically changed without effecting business logic or user interface code.
- The application takes advantage of Java database connectivity (JDBC), minimizing the number of interface points that must be maintained.
- Market-proven and industry-standard technology is utilized (for example, JSPs, JDBC, and so on).

The layering model

The following diagram, together with the explanations that follow, offer a high-level conceptual view of the layers and their responsibilities within the architecture. Key areas of the diagram are described in more detail in the sections that follow.



ReIM's layered architecture

Presentation layer

This area of the architecture encapsulates the graphical user interface (GUI) processing. A web browser accesses JSP pages using a Struts tag library.

JSPs consist of JavaScript and standard HTML. They make calls to tag-libraries. An extension of Java servlet technology, JSPs are compiled into servlets. JSPs provide a user interface that can be separated from most of the business logic that resides on the server. This separation of presentation from content offers a greater possibility for ease of maintenance, both with regard to the page that the user sees and the underlying logic. The look and feel of the GUI is easy to customize, and dynamic functionality is easy to create.

Struts provides an open source framework for building Web applications. The core of Struts is a flexible control layer based upon Java servlets, JavaBeans, ResourceBundles, and Extensible Markup Language (XML). Struts provide an industry standard approach to enforcing the division between user interface code and business logic. Struts also provides standard functionality for error display, internationalization/screen translation, and so on. The Struts framework is part of the Jakarta Project, sponsored by the Apache Software Foundation (<http://www.apache.org/>). The official Struts home page is <http://jakarta.apache.org/struts>.

The presentation layer only interacts with the middle tier services.

Middle tier

Service layer responsible for business logic

The service layer consists of a collection of Java classes that implement business logic (data retrieval, updates, deletions, and so on) via one or more high-level methods. In other words, the service layer controls the workflow. For example, when a user clicks OK on a page, the server must follow a given series of steps to accomplish business functionality. The service layer controls how those steps are accomplished.

The service layer is the entry point to the middle tier and separates the presentation layer from the database layer. Generally the methods that are exposed by service layer classes accept and/or return business objects. The service layer encapsulates the business logic by calling down into business objects and the data access layer, thus making the code more maintainable.

Business objects

Within ReIM, business objects are beans (that is, Java classes that have one or more attributes and corresponding set / get methods) that represent a functional entity. In other words, business objects can be thought of as data containers, which by themselves have almost no business functionality. (In those unusual cases where business logic resides within a business object, the logic pertains to a discreet business concept.) Two examples of business objects include 'Document' and 'Supplier'.

There is not necessarily a one to one relationship between a business object and a database table. The service layer may utilize more than one class from the data access layer in order to combine the data from more than one database table to fully populate a business object.

Data access layer (DAL)

The data access layer interacts only with the middle tier and the database. Classes in the DAL abstract the actual persistence mechanism that is being used to persist business objects. The DAL provides the mechanism that allows ReIM to be associated to a different persistence engine. Ideally, in those cases, only the DAL would need to be modified due to the change. The remainder of ReIM would continue to operate unchanged.

The ReIM DAL consists of two very distinct portions: a DAL to ReIM ‘owned’ tables and an interface DAL to merchandising system tables. The two distinct types of Java code are described below.

Access to invoice matching tables

This code utilizes auto-generated ‘row’ beans, with corresponding ‘access’ classes, one per database table. Both the row and the access class for any given table are automatically generated using a Data Access Layer Generator (DALGen) tool. This tool partly helps mitigate the need for Java developers to manually write SQL code. On occasion, developers may have to ‘extend’ an Access class in order to implement custom SQL that is too specific to be automatically generated. The same tables and database access code is used regardless of the merchandising system’s version. Different versions of the merchandising system’s tables can thus be associated to the application with minimal impact to the code in the middle tier.

Access to merchandising system (such as RMS) tables

Data access ‘driver beans’ are Java classes that are programmed to an abstract factory interface. Each class contains JDBC code that is implemented manually by the developer in order to meet the needs of the middle tier business logic. For a definition of JDBC, see the “ReIM-related Java terms and standards” section in this chapter.

The abstract factory design pattern allows for different versions of the merchandising tables to be associated to the application with minimal impact to the code in the middle tier.

Database layer

The database layer is the application’s storage platform, containing the physical data (user and system) used throughout the application. This layer is only intended to deal with the storage and retrieval of information and is not involved in the manipulation of the data.

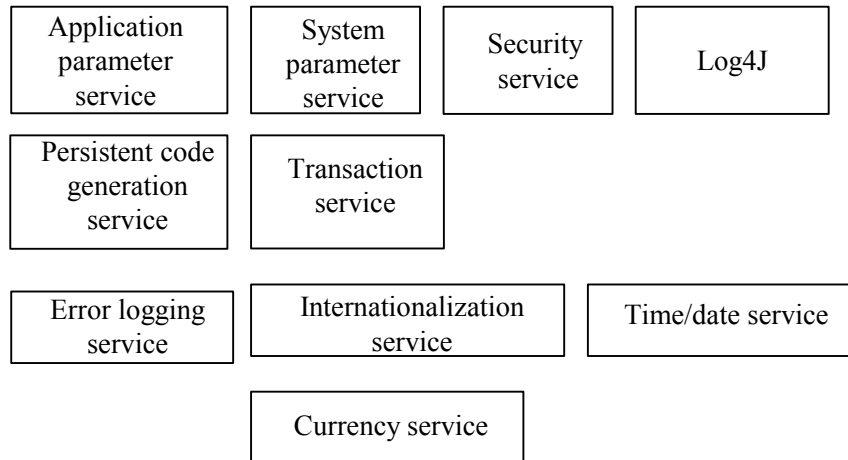
Technical services

In order to increase the maintainability of the code, and enhance the rapid development of new business logic, a number of base technical services are provided.

Technical services hold the application together by providing common services to the application, services that are not necessarily driven by business requirements.

Technical services include application frameworks such as error logging, internationalization, transaction management, application security, and so on.

A brief description of each technical service follows the diagram.



ReIM's technical services

Application parameter service.

This service allows application configuration parameters to be stored within the database on a single database table. Developers can retrieve these parameters using a high level interface.

System parameter service

Similar to the application parameter service, this service is used only for technical configuration parameters. Although most configurable parameters are hosted in a system parameter table, some parameters are located in a properties file. See Chapter 2, “Initial implementation and system parameters”.

Transaction service

Note: The transaction service does not provide checkpoint transaction management or multi-phase commit.

This service provides a simplified management of rollback/commit semantics. In order to avoid the need to pass the database connection between the middle tier method calls and the data access layer classes, the transaction service uses thread local variables to maintain the current connection for a thread until that thread has committed or rolled back the transaction. This service thus simplifies transaction management.

Persistent code generation service

This technical service consists of the Data Access Layer Generator (DALGen) tool. Having executed a number of queries against the Oracle metadata tables, this tool can generate generic JDBC code for accessing a specific table. DALGen generates a class for any given database table, based on a configuration file. The bean class is capable of the basic database operations: create, insert, update, and delete. As applicable, there are both bulk and single row versions of each database operation. In addition to promoting rapid application development, automated code generation can make broad sweeping changes to the manner in which the ReIM application makes JDBC calls. Those changes can be rolled out without an expensive re-engineering effort.

Error logging service

This service incorporates a standard ReIMException class to raise and handle Java exceptions (shown below). The ReIMException class automatically logs itself to the application log file. The level of logging may be raised or lowered in the properties file. For example, an operator could configure the system to only display INFO and above. See Chapter 2, “Initial implementation and system parameters”.

- FATAL
- ERROR
- WARN
- INFO
- DEBUG
- PERFORMANCE

The system’s coding pattern ensures that the error messages, no matter where they originate, remain detailed in their presentation to the operator. For example, if a business specific message is thrown near the database, such as that an item-supplier combination does not exist, the system does *not* genericize that exception under a ‘could not post transaction’ message or something similar. Rather, the error message is presented in all of its original detail for the operator.

Log4J

This service provides the error logging services with a standard method for logging information to a flat text file. Log4J is an open source product.

Internationalization service

This service uses resource files to provide configurability for on-screen messages (such as on screen labels or error messages). To change the language for the ReIM GUI screens, a replacement set of resource files can be created. Note that although this service supports any number of languages, the screen flow remains left to right, top to bottom.

Currency service

This service provides a high-level mechanism for developers to represent a currency amount. This service provides the formatted representation of that currency.

Time/date service

This service provides a high level interface to the Java time/date constructs along with some formatting methods for displaying these constructs on the GUI screens.

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. The physical implementation of the security information for each user, role, functional authorizations, and field authorizations is independently configurable among the database or LDAP server locations.

Third party libraries

ReIM base development uses the following third party libraries:

- Oracle JDBC 2.0 library
- Log4J
- Junit from www.junit.org
- Struts from jakarta.apache.org

ReIM-related Java terms and standards

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

The Java 2 Enterprise Edition (J2EE)

The Java standard infrastructure for developing and deploying multi-tier applications. Implementations of J2EE provide enterprise-level infrastructure tools that enable such important features as database access, client-server connectivity, distributed transaction management, and security.

Java Database Connection (JDBC)

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

JDBC provides the support that allows ReIM to submit SQL queries to the database and receive the result set for further processing..

Java Development Kit (JDK)

Standard Java development tools from Sun Microsystems.

Java Server Pages (JSP)

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

Java Servlet

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

LOG4J

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

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. The first letter of each internal word is capitalized.
- Interfaces: These descriptive names are unabbreviated nouns that have both lower and upper case letters. The first letter of each internal word is capitalized.
- Methods: Methods begin with a lowercased verb. The first letter of each internal word is capitalized.

Struts, version 1.0.2

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

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

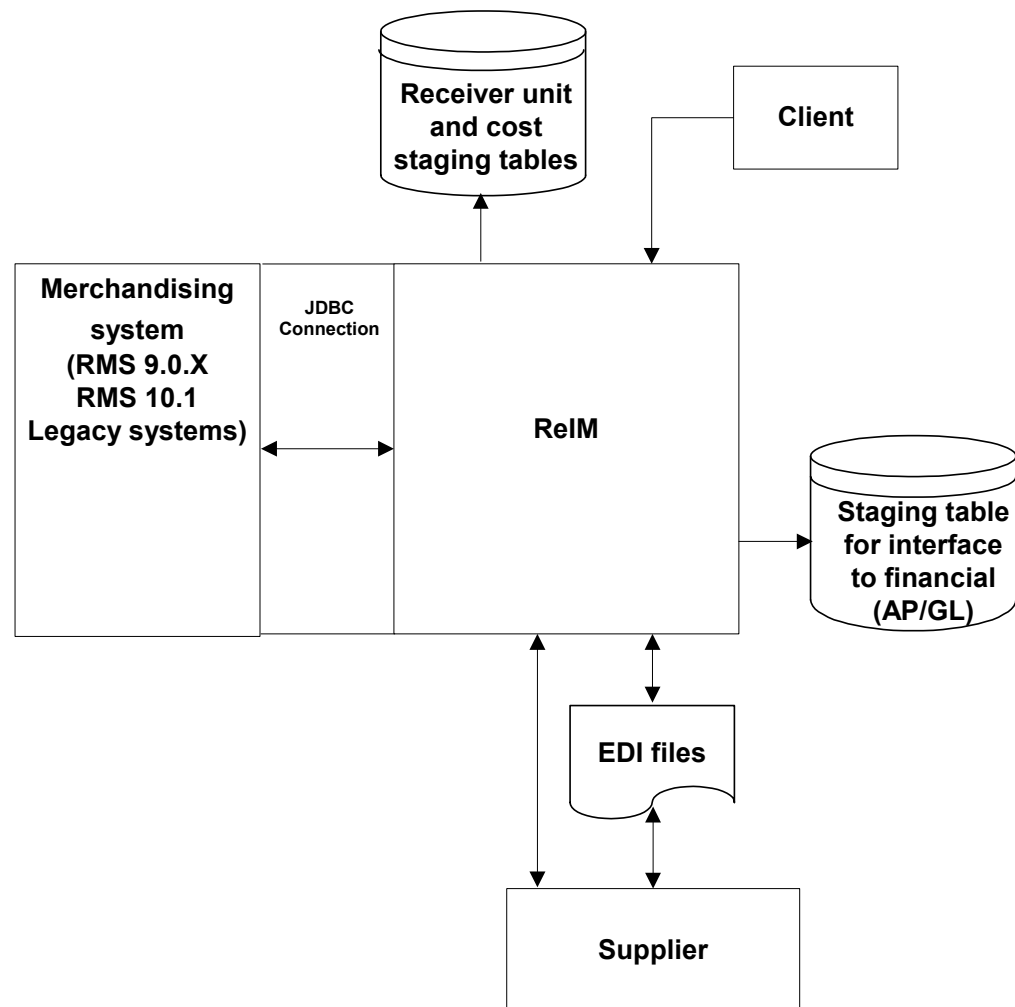
Chapter 4 – Functional design

This chapter provides the following:

- An overview as to how ReIM is functionally integrated with other systems (including other Retek systems). The discussion primarily concerns the flow of ReIM-related business data across the enterprise.
- An explanatory discussion of the auto-match process.
- An explanatory discussion of best terms calculations.

Dataflow overview

This section provides you with a diagram that shows the overall direction of the data among the applications and tables. The accompanying explanations are written from a system/staging table-to-system/staging table perspective, illustrating the movement of data.



ReIM dataflow across the enterprise

From the supplier (to EDI) to ReIM

- External invoices via EDI
ReIM requires EDI documents to conform to an ReIM file standard. ReIM assumes that the client, in prior processing, has formatted the files received from suppliers. ReIM processing reads the file(s) and inserts the records into the applicable ReIM tables.

For more information about the ReIM and EDI interface, see Chapter 5, “Interfaces and file layouts”.
- External invoices *not* sent via EDI
ReIM can receive invoices directly from a supplier through methods other than EDI. Information from these types of invoices (paper, for example) must be entered into ReIM manually.
- Credit notes
A response from the supplier indicating agreement to the terms outlined in a credit note request.

From ReIM (to EDI) to the supplier

ReIM processing reads records from ReIM tables and creates a file of documents. ReIM produces document files according to an ReIM file standard.

- Debit memos
A document created to charge a supplier an over-invoiced amount resulting from a price or quantity discrepancy.
- Credit memos
A document created to pay back a supplier for an under-invoiced amount.
- Credit note requests
A document that is sent from the retailer to the supplier, requesting a credit note for an over invoiced amount.

From ReIM to the financial (AP/GL) staging table

ReIM exports data to a financial staging table. The client must create its own interface to deliver this information to the applicable financial system.

- Matched invoices and approved documents
Invoices can be matched through auto-matching or manual matching. The unit cost and quantities of all items on the invoice are compared to the unit cost and quantities on the receipt. If the cost and quantity on the invoice and receipt match within the tolerances defined, there is a match.
 - Debit memos
 - Credit memos
 - Credit notes
- Pre-paid invoices
The process of sending an invoice to accounts payable without matching it to any receipts.

- Non-merchandise invoices
Bills for non-merchandise costs only. Non-merchandise invoices can not contain items. Either suppliers or partners can create non-merchandise invoices.
- Posting transaction codes
Through the front end, users can associate specific external account IDs to each type of financial transaction. Transaction codes represent the type of transaction recorded. Transaction codes include the following:
 - Trade accounts payable
 - Unmatched receipt
 - Variance within tolerance
 - Pre-paid asset
 - Credit note
 - Receipt write off
 - Reason-code driven transactions
 - Non-merchandise code transactions

From the client to ReIM

- Terms ranking file
A term defines the discount that applies if the invoice is paid early and the number of days until payment is due. Terms are the payment conditions negotiated between suppliers and retailers. Terms are associated with suppliers, purchase orders, invoices and other documents. The terms ranking table (IM_TERMS_RANKING) stores the rankings. A flat file is used to populate the table with the terms ID and the terms ranking. The client is responsible for creating the flat file in the prescribed format. The terms ranking files should be uploaded to ReIM on a periodic basis. ReIM processing writes the terms ranking data to the ReIM tables.

From the merchandising system to ReIM

Note for RMS users only:

Item, purchase order, supplier, and other data are accessed directly from the RMS tables, with no need to interface data via batch modules.

Via a Java database connectivity (JDBC) link, ReIM receives the following data:

- Non-merchandise codes
- Currency
- Exchange rates
To support the invoice matching process, the merchandising system provides exchange rates data.
- Store/warehouse
- Supplier

- Supplier address (Invoice address, Returns address, and so on)
- Supplier trait

In the merchandising system, supplier traits represent a grouping mechanism for suppliers with common characteristics. They are utilized for mass updates. This data is used in setting up tolerances within ReIM.
- Merchandise hierarchy
- Item data

ReIM processes only transaction-level items (SKUs), so there is no need to interface parent (or style) level items. However, orderable pack items must be interfaced because they can exist on an invoice/receipt. Also, for reference purposes UPCs may be used so they should be provided by the merchandising system. See the RMS documentation for more information about its three-level item structure.
- Receipts

A receipt is a document stating that the goods have arrived to the store or warehouse. For the purposes of ReIM, only goods associated with purchase order (as opposed to transfers) are considered. Receipt data is interfaced from RMS and stored within ReIM because ReIM-specific actions are performed on receipts within the product (for example, trimming receipt quantities and creating new overage receipts, updating statuses, and so on).
- Purchase order data

An agreement between a retailer and a supplier for the purchase of goods. The retailer records the quantity, cost, and delivery location of items from the supplier. On a single purchase order, the same item going to different locations can have different costs. Purchase order data is integral to invoice matching processes.
- Business date
- Partner

A partner is a business that supplies and bills a client for non-merchandise services. Examples of partners are banks, agents, and expense suppliers. A partner cannot send merchandise invoices to clients.
- Terms

A term defines the discount that applies if the invoice is paid early and the number of days until payment is due. Terms are the payment conditions negotiated between suppliers and retailers. Terms are associated with suppliers, purchase orders, invoices and other documents.

From ReIM to receiver unit and cost staging tables

- Receiver cost adjustment
The client is responsible for creating a process that uses the receiver cost adjustment data. An automated process may perform the following with the data:
 - Update order cost
 - Update receipt cost
 - Update supplier cost
 - Update open to buy (OTB), weighted average cost (WAC), and so on
- Receiver unit adjustment
The client is responsible for creating a process that uses the receiver cost adjustment data. An automated process may perform the following actions with the data:
 - Update receipt quantity
 - Receive additional units
 - Update weighted average cost (WAC), and so on

From ReIM to the merchandising system

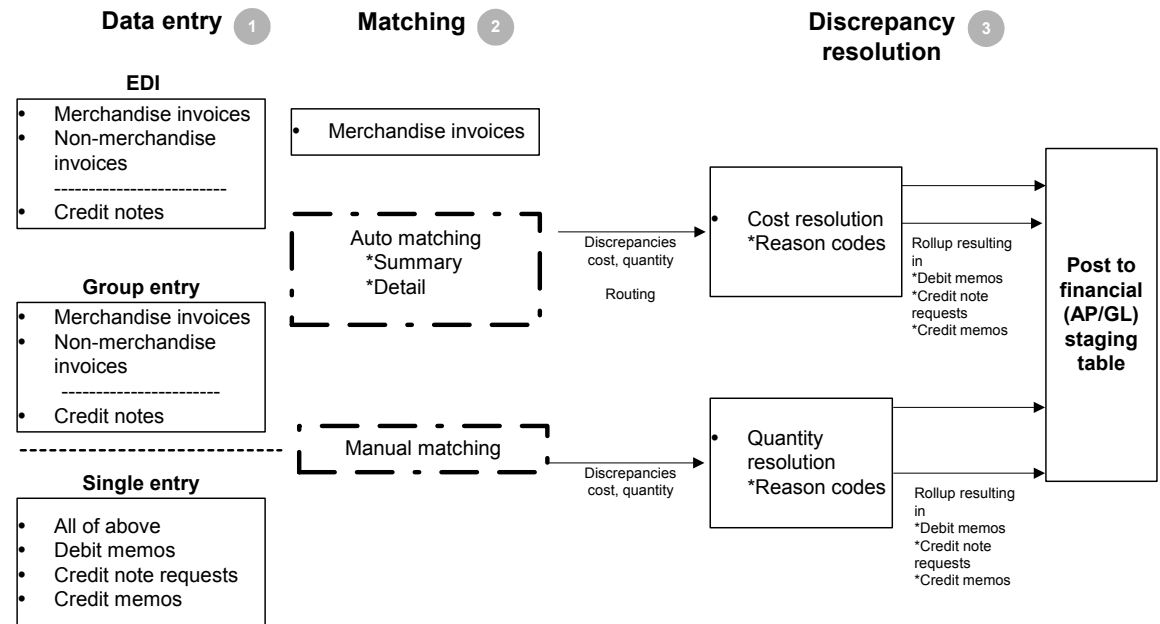
- Receipt status

Invoice matching process flow

This section provides a high-level explanation of the invoice matching process flow for each of the following areas:

- Data entry
- Matching
- Discrepancy resolution

Explanations of each numbered item on the diagram follow it.



High-level view of the invoice matching process

Note: Documents ‘drop out’ of the flow when they need no further processing. For example if an invoice is matched in step 2, ‘Matching’, the document would not continue to step 3, ‘Discrepancy resolution’. The document would be posted directly to the financial (AP/GL) staging table after step 2.

- 1 Data entry

There are three ways in which invoices and other documents enter the ReIM system:

 - EDI

EDI allows ReIM to upload all of the following:

 - Merchandise invoices

The bill for goods or services received from a supplier or partner. Merchandise invoices may have both of the following:

 - Merchandise costs

Costs that are associated with items on documents. Any other costs on an invoice are non-merchandise costs. The sum of the merchandise costs and non-merchandise costs is the total document cost.

- Non-merchandise costs (shipping and handling, and so on)
Costs that are not associated with items, such as shipping charges.
- Non-merchandise invoices
Bills for non-merchandise costs only (a snow shoveling service for example). Non-merchandise invoices cannot contain items. Either suppliers or partners can create non-merchandise invoices.
- Credit notes
A response from the supplier indicating agreement to the terms outlined in a credit note request, which is a document that is sent from the retailer to the supplier, requesting a credit note for an over invoiced amount. Note that, compared to invoices, credit notes represent a separate entity in the functional process flow.
- Group entry (through the front end)
Group entry facilitates the mass entry of the same type of documents that enter the system through EDI, but in the case of group entry, the documents are paper rather than electronic.
- Single entry (through the front end)
Single entry is designed as an exception handling tool made for invoices that do not enter (for whatever reason) within a group. Single entry facilitates the entry of the same type of documents that enter the system through EDI. Single entry also allows the client to create the following, if they have not been created through other processes:
 - Debit memos
A document created to charge a supplier an over invoiced amount resulting from a price or quantity discrepancy. A debit memo immediately removes funds from a supplier's account.
 - Credit note requests
A document that is sent from the client to the supplier, requesting a credit note for an over invoiced amount. A credit note request informs the supplier that if the client does *not* receive a credit note for the applicable funds, the funds will be removed from a supplier's account.
 - Credit memos
A document created to pay back a supplier for an under-invoiced amount.

2 Matching

This step in the process flow handles only merchandising invoices. Other documents (credit notes, debit memos, and so on) are sent directly to the financial (AP/GL) staging table.

- Auto-matching

Merchandise invoices are grouped by their PO/location; all merchandise invoices have these two attributes. The system accesses the merchandising system to determine what shipments were created for that PO/location. The shipment data gives the system an indication of goods received. The invoices represent the goods for which the client is being billed. The system examines the difference between these values.

If auto-match is able to find differences that are out of the pre-established tolerance range, the system creates discrepancies (cost or quantity). Otherwise, matched invoices are posted to the financial (AP/GL) staging table.

For more functional information about summary and detail-level auto-matching, see ‘The auto-match process’ section later in this chapter.

- Manual matching

Through manual matching, users can perform the same type of matching transactions that occur in the auto-match process but with greater flexibility. Invoices are initially grouped by their PO/location, but the groups can be changed, other receipts can be brought in, and so on.

Manual match either matches a document, which is posted to the financial (AP/GL) staging table, or manual match creates a discrepancy (cost or quantity).

3 Discrepancy resolution

Pre-established routing rules determine which users receive which discrepancies (for example, a store manager might review quantity discrepancies; a buyer might review cost discrepancies, and so on).

Documents are resolved by being associated to reason codes. Depending upon the specific reason code, one of the following two outcomes may result:

- The document is posted to the financial (AP/GL) staging table.
- A debit memo, a credit note request, and/or a credit memo is generated and, if applicable, posted.

The auto-match process

Overview

Invoices in ready for match, unresolved, and multi-unresolved status are retrieved from the database to be run through the auto-match algorithm. These invoices are grouped with receipts based upon PO location.

If no receipts exist for the PO location, the invoices are sent to the cost pre-matching algorithm.

If receipts do exist, the system attempts to make a match for all invoices and receipts for the PO location.

If that attempt fails, the system attempts to match each invoice to a single receipt in the one-to-one matching algorithm. If all invoices are matched in this fashion, then the next PO location is processed.

If, all of the invoices cannot be matched and a multi-unresolved scenario results, the matched invoices remain matched, and the non-matched invoices are all given a multi-unresolved status. No further processing occurs for this PO location.

If one unmatched invoice is eligible for line level matching, an attempt is made to match each line on the invoice to an unmatched receipt line.

This section describes the following functionality related to the auto-match process:

- Cost pre-matching
- PO location summary matching
- One-to-one invoice matching
- One-to-one invoice matching
- Eligibility for line level matching
- Line level matching
- Recycling and overall flow
- Partially matched receipts
- Tolerance
- History and metrics

Cost pre-matching

Cost pre-matching occurs only for PO locations that meet the following conditions:

- Invoices that have never been processed by auto-match exist.
- No receipts exist.

Each invoice line unit cost is compared with the PO item location's unit cost. If the unit costs match within tolerance, the invoice and lines are processed again by auto-match once receipts come in for the PO location.

If there is a discrepancy, then the invoice is processed again once receipts arrive. However, the lines that contain a discrepancy are immediately routed for cost resolution. Once invoices are run through the cost pre-matching algorithm, they are *not* re-run when the next auto-match run occurs if there are still no receipts.

Scenarios can arise where no receipt lines exist, and no order line corresponds to an invoice line. The assumption is that validation occurs in the EDI upload process and in the manual invoice entry screens prevent these invoices from entering the system. Therefore, auto-match ignores this situation.

PO location summary group matching

PO location summary group matching processes the following:

- Invoices that have never been processed before by auto-match.
- Invoices that have been processed previously by auto-match but remain unresolved.
- Invoices that have been processed previously by auto-match but that have been identified as multi-unresolved.

First, the systems attempts to match the total extended cost of the invoices with the total extended cost of the receipts. Extended cost is defined as the unit cost for an item multiplied by the quantity received or the quantity invoiced. For this comparison, all extended costs are summed for the group of invoices and receipts and compared. The total extended cost for each invoice is taken from the invoice header. The process, however, calculates the receipts' total extended cost.

Quantity matching is also sometimes required. Whether or not quantity matching is performed is determined by a supplier option. Quantity matching compares the total quantity invoiced for the PO location with the total quantity received for the PO location. As in cost matching, the total quantity invoiced for each invoice is taken from the invoice header. For receipts, the process calculates this sum.

Auto-match processing first attempts to match the total extended costs, and optionally the total quantities, exactly. If the costs and quantities do not match exactly, then the system attempts to match them within tolerance. If a match is achieved, all of the invoices, receipts, and their lines for the PO location are assumed to be matched. If a match is *not* achieved, all invoices and receipts for the PO location are unresolved. These invoices and receipts are processed further with one-to-one invoice matching.

Auto-match accounts for the actions taken by cost reviewers that fully resolve a cost discrepancy when attempting to match at the summary level. If a match is achieved at the summary level, auto-match deletes any outstanding unresolved cost discrepancies and any partially resolved cost discrepancies along with their partial resolutions for the PO location from the system.

Example 1

The following example illustrates a successful match:

Invoices for a PO location	Total Extended Cost	Total Quantity
Invoice 1	\$50,000	1000
Invoice 2	\$150, 000	5000
Totals:	\$200,000	6000

Receipts for a PO location	Total Extended Cost	Total Quantity
Receipt 1	\$50,000	2000
Receipt 2	\$50,000	2000
Receipt 3	\$100,000	2000
Totals:	\$200,000	6000

In the example, the total extended costs and the total quantities match for the PO location. Therefore, all invoices and receipts will be set to matched status.

Example 2

The following example illustrates a successful match, but where quantity matching is not required by the supplier.

Invoices for a PO location	Total Extended Cost	Total Quantity
Invoice 1	\$50,000	2000
Invoice 2	\$150, 000	5000
Totals:	\$200,000	7000

Receipts for a PO location	Total Extended Cost	Total Quantity
Receipt 1	\$50,000	2000
Receipt 2	\$50, 000	2000
Receipt 3	\$100,000	2000
Totals:	\$200,000	6000

In the example, only the total extended costs match. However, quantity matching is not required for this supplier. Therefore, these invoices and receipts are considered matched by the auto-matching algorithm.

Example 3

The following example illustrates an unsuccessful match, where quantity matching is required by the supplier.

Invoices for a PO location	Total Extended Cost	Total Quantity
Invoice 1	\$50,000	1000
Invoice 2	\$150, 000	5500
Totals:	\$200,000	6500

Receipts for a PO location	Total Extended Cost	Total Quantity
Receipt 1	\$50,000	2000
Receipt 2	\$50, 000	2000
Receipt 3	\$100,000	2000
Totals:	\$200,000	6000

In the example, because quantity matching is required for the supplier, the match is unsuccessful despite the fact that the costs do match. The invoices and receipts will be set to unresolved status, and an attempt will be made to match them at a one-to-one level.

Example 4

The following example illustrates a successful match within tolerance.

Invoices for a PO location	Total Extended Cost	Total Quantity
Invoice 1	\$50,035	1000
Invoice 2	\$150,100	5000
Totals:	\$200,135	6000

Receipts for a PO location	Total Extended Cost	Total Quantity
Receipt 1	\$50,000	2000
Receipt 2	\$50,000	2000
Receipt 3	\$100,000	2000
Totals:	\$200,000	6000

In the example, even though there is not an exact match, the extended costs match within tolerance. Therefore, the receipts and invoices are set to matched status.

One-to-one invoice matching

One-to-one invoice matching attempts to match each invoice for the PO location with a single receipt for the PO location. First, the system attempts a match between the total extended costs. If the extended costs match, the system may, depending upon a supplier option, attempt a match between the total quantities. If there is either an exact match or a match within tolerance, the invoice and receipt along with their lines are considered to be matched. If no match can be found for the invoice, it is left unresolved.

One-to-one matching may result in a multi-unresolved scenario. If any invoices within the PO location can be successfully matched with *one and only one* receipt and that receipt can be matched to *only one* invoice for the PO location, then those invoices and receipts are considered to be matched. If no unmatched invoices remain, then processing stops for the PO location and all invoices are considered matched. Only when one unmatched invoice exists for the PO location can line level matching occur. If more than one invoice remains after one-to-one matching, then all remaining unmatched invoices and receipts are considered to be multi-unresolved.

One-to-one matching is driven by invoices. Therefore, if there are unmatched receipts remaining but no unmatched invoices for the PO location, no further processing occurs. The receipts remain unresolved but no discrepancies are generated.

Example 1

The following example illustrates how one invoice matches with one and only one receipt. One invoice and two receipts are unresolved.

Invoices for a PO location	Total Extended Cost	Total Quantity	Status post matching
Invoice 1	\$50,000	5000	Matched
Invoice 2	\$100,000	10,000	Unresolved

Receipts for a PO location	Total Extended Cost	Total Quantity	Status post matching
Receipt 1	\$50,000	5000	Matched
Receipt 2	\$25,000	2500	Unresolved
Receipt 3	\$35,000	2500	Unresolved

In the example, Invoice 1 matches with Receipt 1. However, the remaining invoice and receipts do not match one-to-one. Because there are two unmatched receipts remaining and only one unmatched invoice, the remaining unmatched invoice and receipts are considered to be unresolved. If they are eligible for detail matching, they are sent to the detail-matching algorithm.

Example 2

The following example illustrates a multi-unresolved match, with no successful matches.

Invoices for a PO Location	Total Extended Cost	Total Quantity	Status Post Matching
Invoice 1	\$50,000	5000	Multi-unresolved
Invoice 2	\$25,000	2500	Multi-unresolved
Invoice 3	\$35,000	3000	Multi-unresolved

Receipts for a PO Location	Total Extended Cost	Total Quantity	Status Post Matching
Receipt 1	\$40,000	4000	Multi-unresolved
Receipt 2	\$25,000	2500	Multi-unresolved
Receipt 3	\$25,000	2500	Multi-unresolved
Receipt 4	\$10,000	1000	Multi-unresolved

In the example, Invoice 2 can be successfully matched to both Receipt 2 and Receipt 3. Therefore, no match can be obtained for Invoice 2. All invoices and receipts are set to multi-unresolved status.

Example 3

The following example illustrates another multi-unresolved match, with no successful matches.

Invoices for a PO location	Total Extended Cost	Total Quantity	Status post matching
Invoice 1	\$40,000	4000	Multi-unresolved
Invoice 2	\$25,000	2500	Multi-unresolved
Invoice 3	\$25,000	2500	Multi-unresolved
Invoice 4	\$10,000	1000	Multi-unresolved

Receipts for a PO location	Total Extended Cost	Total Quantity	Status Post Matching
Receipt 1	\$50,000	5000	Multi-unresolved
Receipt 2	\$25,000	2500	Multi-unresolved
Receipt 3	\$35,000	3000	Multi-unresolved

In the example, Receipt 2 can be successfully matched to both Invoice 2 and Invoice 3. All invoices and receipts are set to multi-unresolved status.

Example 4

The following example illustrates a multi-unresolved match, but one with successful matches.

Invoices for a PO location	Total Extended Cost	Total Quantity	Status Post Matching
Invoice 1	\$50,000	5000	Multi-unresolved
Invoice 2	\$25,000	2500	Multi-unresolved
Invoice 3	\$35,000	3000	Matched

Receipts for a PO Location	Total Extended Cost	Total Quantity	Status Post Matching
Receipt 1	\$40,000	4000	Multi-unresolved
Receipt 2	\$25,000	2500	Multi-unresolved
Receipt 3	\$25,000	2500	Multi-unresolved
Receipt 4	\$35,000	3000	Matched

In the example, Invoice 2 can be successfully matched with both Receipt 2 and Receipt 3. Invoice 3, however, can be successfully matched only with Receipt 4. Therefore, Invoice 3 and Receipt 4 are set to matched status. All other invoices and receipts for the PO location are set to multi-unresolved status.

Example 5

The following example illustrates a scenario in which all invoices match, but there are remaining unresolved receipts.

Invoices for a PO Location	Total Extended Cost	Total Quantity	Status Post Matching
Invoice 1	\$50,000	5000	Matched
Invoice 2	\$25,000	2500	Matched
Invoice 3	\$35,000	3000	Matched

Receipts for a PO Location	Total Extended Cost	Total Quantity	Status Post Matching
Receipt 1	\$50,000	5000	Matched
Receipt 2	\$25,000	2500	Matched
Receipt 3	\$15,000	2500	Unresolved
Receipt 4	\$35,000	3000	Matched
Receipt 5	\$75,000	10,000	Unresolved

In the example, all three invoices can be successfully matched to one and only one receipt. However, two unmatched receipts remain. The invoices are still considered matched, and the receipts remain unresolved.

Eligibility for line level matching

In auto-matching, matching can be performed for entire invoices or broken down to the line level. PO location level matching and one-to-one invoice matching are performed for entire invoices and receipts. Line level matching is performed by item.

In order to be eligible for line level matching, an invoice or receipt must meet the following conditions:

- 1 Neither the invoice nor receipt can be in multi-unresolved status.
If the invoice or receipt is in multi-unresolved status, it is assumed that human intervention is required. No further attempts are made to match the applicable invoice at the line level.
- 2 Lines must be present on the invoice.
Auto-matching assumes that invoices either have all lines in the system or no lines. The system neither validates nor processes partial invoices. If any lines are present, auto-matching assumes that all lines are present.

- 3 The number of days to routing must be exceeded.
The system uses settings and a formula to arrive at its determination of routing days. A supplier option is used to define how long the system should wait before routing discrepancies for invoices for that supplier. However, if the invoice is due sooner than the routing date, then discrepancies may be routed earlier than the route date. There is a system option that determines the maximum number of days before an invoice's due date that discrepancies for an invoice must be routed. The earliest date between the routing date defined by the supplier option and the routing date dictated by the system option is the date on which auto-match routes discrepancies for an invoice.

Supplier option: routing days = x days

System option: maximum days before due date = y days

Supplier driven routing date = invoice date + x days

System driven routing date = invoice due date – y days

The actual routing date equals the earlier of the supplier driven routing date and the system-driven routing date.

Line-level matching

If only one invoice remains unmatched and zero-to-many receipts are unmatched for the PO location and the invoice is eligible, the system attempts to match each line item on the invoice to receipt line items on the receipts for the same item. If a match is not found, price and/or quantity discrepancies are created and routed. Once line level matching is complete for a PO location, if all lines have been matched, then the entire invoice and all of the receipts are considered matched. Otherwise, they remain unresolved.

When invoice lines are sent through line level matching, all existing unresolved or partially resolved cost discrepancies are deleted along with any partial resolutions. If line level matching produces new discrepancies, they are created and routed, thus ensuring that discrepancies are routed with the latest information available about the invoice and receipt lines.

If no receipt lines correspond to an invoice line, cost pre-matching is attempted for the applicable invoice line using the PO's unit cost. The system assumes that the invoice line exists on the order. If there is a discrepancy, a cost discrepancy is created and routed. In this scenario, a quantity discrepancy is automatically created and routed where the entire invoiced quantity is the discrepant quantity.

For line level matching, cost and quantity matching are always performed. Beyond that, if cost matching fails, quantity matching is still performed in order to route potential quantity discrepancies that may be discovered. When discrepancies are created, the PO's supplier is associated with the discrepancy.

For quantity line level matching, the comparison is made between the quantity invoiced and the sum of the quantities received across the receipts for that item. If a quantity match cannot be obtained, then a quantity discrepancy is generated and routed for the invoice line and the receipt line(s) for that item.

Example 1

The following example illustrates a scenario in which all lines match, and the invoices and receipts are set to matched status.

Invoice Lines for a PO location	Item	Unit Cost	Quantity	Status post matching
Invoice 1			550	Match
- Invoice line	Item 1	\$5000	100	Match
- Invoice line	Item 2	\$10,000	200	Match
- Invoice line	Item 3	\$15,000	250	Match

Receipt Lines for a PO location	Item	Unit Cost	Quantity	Status post matching
Receipt 1			565	Match
- Receipt line	Item 1	\$5020	105	Match
- Receipt line	Item 2	\$10,100	210	Match
- Receipt line	Item 3	\$15,030	250	Match

In the example, all lines match *within tolerance*. The lines' statuses are set to matched. Because all lines on the invoice and receipt match, the invoice and receipt statuses are set to matched.

Example 2

The following example illustrates a scenario in which some lines match, and the invoices and receipts are remain in unresolved status.

Invoice Lines for a PO location	Item	Unit Cost	Quantity	Status post matching
Invoice 1			550	Unresolved
- Invoice line	Item 1	\$12,000	100	Unresolved
- Invoice line	Item 2	\$10,000	200	Match
- Invoice line	Item 3	\$12,000	250	Unresolved

Receipt Lines for a PO location	Item	Unit Cost	Quantity	Status post matching
Receipt 1			550	Unresolved
- Receipt line	Item 1	\$5000	100	Unresolved
- Receipt line	Item 2	\$10,000	200	Match
- Receipt line	Item 3	\$10,000	250	Unresolved

In the example, the lines value for Item 2 is matched. However, because Items 1 and 3 do *not* match, the receipt and invoice are unmatched.

Example 3

The following example illustrates a scenario in which some lines match, and the invoices and receipts remain in unresolved status. Note that one invoice line has no corresponding receipt item.

Invoice Lines for a PO location	Item	Unit Cost	Quantity	Status post matching
Invoice 1			550	Unresolved
- Invoice line	Item 1	\$12,000	100	Unresolved
- Invoice line	Item 2	\$10,000	200	Match
- Invoice line	Item 3	\$12,000	250	Unresolved

Receipt Lines for a PO location	Item	Unit Cost	Quantity	Status post matching
Receipt 1			550	Unresolved
- Receipt line	Item 1	\$5000	100	Unresolved
- Receipt line	Item 2	\$10,000	200	Match

Order Lines for a PO location	Item	Unit Cost
- Order line	Item 1	\$5000
- Order line	Item 2	\$10,000
- Order line	Item 3	\$12,000

In the example, Item 2 matches. A cost discrepancy is created for Item 1. No cost discrepancy is created for Item 3 because its unit cost matches the PO's unit cost. A quantity discrepancy is created for Item 3 where the received quantity is zero because the item is not on the receipt.

Example 4

The following example illustrates a scenario in which one invoice line is matched to many receipt lines.

Invoice lines for a PO location	Item	Unit cost	Quantity	Status post matching
Invoice 1				Match
- Invoice line	Item 1	\$5	100	Match

Receipt lines for a PO location	Item	Unit cost	Quantity	Status post matching
Receipt 1				Match
- Receipt line	Item 1	\$5	70	Match
Receipt 2				Match
- Receipt line	Item 1	\$5	30	Match

In the example, one invoice line can be matched with two receipt lines.

Recycling and overall flow

As soon as invoices arrive, auto-matching processes them. If there are no receipts, the invoices are sent to cost pre-matching immediately where cost discrepancies are created and routed if needed.

Once receipts arrive, the invoices and receipts are matched at the PO location level and at the one-to-one level. If no match exists, these invoices and receipts are recycled through summary level matching until the routing days has passed. If there is a match, then any unresolved or partially resolved cost discrepancies are removed from the system.

For discrepancies that have been fully resolved, the actions taken are reflected in the adjusted total extended cost and adjusted total quantity of the invoices and the receipts. Therefore, summary level matching will respond to human intervention and match invoices that become matched after discrepancy resolution.

After the routing days have passed, an invoice for a PO location that remains unmatched undergoes line level matching. In this type of scenario, all existing unresolved or partially resolved cost discrepancies are deleted. New cost and quantity discrepancies are created if any exist.

After line level matching is performed for an invoice (through either the auto-match or the manual matching process), that invoice is never processed by auto-match again.

Partially matched receipts

Partially matched receipts are eligible for matching with invoices that may arrive at a later date. At a summary level, unmatched lines are totaled and compared with the total extended cost and total quantity, if required, for the unprocessed invoice or invoices. If there is a match, the invoice(s) are considered matched and the receipt is now completely matched.

From the front end, users may choose to ‘split’ a receipt item. That is, they use some of the receipt line’s quantity to create a match or resolve a discrepancy online. Auto-match recognizes that some of a receipt line item’s quantity has been used and only uses the unmatched or available portion of the quantity to perform matching.

Example 1

The following example illustrates summary level matching:

Invoice lines for a PO location	Item	Extended Cost	Quantity	Status prior to matching	Status after matching
Invoice 1		\$30,000	500	Unresolved	Matched

Receipt lines for a PO location	Item	Extended Cost	Quantity	Status prior to matching	Status after matching
Receipt 1		\$60,000	1000	Partially Matched	Matched
- Receipt line	Item 1	\$30,000	500	Previously matched	Matched
- Receipt line	Item 2	\$15,000	250	Unresolved	Matched
- Receipt line	Item 3	\$15,000	250	Unresolved	Matched

In the example, a partially matched receipt is used to match an unprocessed invoice. Only the unmatched lines for the receipt are used to determine whether the invoice and receipt match at the summary level.

Example 2

The following example illustrates line level matching:

Invoice lines for a PO location	Item	Unit Cost	Quantity	Status prior to matching	Status after matching
Invoice 1				Unresolved	Unresolved
- Invoice line	Item 2	\$15,000	250	Unresolved	Unresolved
- Invoice line	Item 3	\$15,000	250	Unresolved	Matched

Receipt lines for a PO location	Item	Unit Cost	Quantity	Status prior to matching	Status after matching
Receipt 1				Partially Matched	Matched
- Receipt line	Item 1	\$30,000	500	Previously matched	Matched
- Receipt line	Item 2	\$15,000	250	Previously matched	Matched
- Receipt line	Item 3	\$15,000	250	Unresolved	Matched

In the example, the invoice remains unresolved and the receipt becomes matched. Even though Item 2 of Invoice 1 matches with Item 2 of Receipt 1, Item 2 of Receipt 1 had already been matched to a different line on a different invoice. Therefore, it is not reused here to make a match. Item 3 of Receipt 1 is unresolved and is therefore available to be matched to Item 3 of Invoice 1.

Tolerance

When selecting the correct tolerance to apply to a particular match, the following three important parameters should be considered:

- 1 Is the match at the summary or line level?
- 2 Is the match for cost or quantity?
- 3 Is the discrepancy in favor of the retailer or the supplier?

The first parameter refers to whether the match being attempted is some sort of summation of costs and/or quantities or to whether the match is for a single line item. PO location level matching and one-to-one invoice matching are types of summary matching because they involve comparing aggregate costs and quantities. Line level matching, however, refers only to a single line item's unit cost and quantity.

The second parameter refers to whether the tolerance is needed for a comparison of costs or for a comparison between quantities.

The third parameter refers to whether the discrepancy is in favor of the retailer or in favor of the supplier. For example, if the invoice cost is 20 and the purchase order cost is 30, the discrepancy of 10 is in favor of the retailer because the invoice cost is less than what was expected.

Tolerances may be defined at the supplier level, the department level, or the corporate level.

The auto-match process attempts to always retrieve the tolerance for the PO's supplier first. If supplier level tolerances do *not* exist, the system's processing moves to the department. If line level matching is being performed, then there is only one item and that item's department is used to retrieve the tolerances. If summary level matching is being performed, then an item is selected at random and that item's department is used to retrieve tolerances. If no department level tolerances exist, the system uses system-level tolerances.

History and metrics

The system records summary and detail history for successful matches that it finds. In addition, the system records whether or not each match was exact or not. At the summary level, the set of receipts that match to a set of invoices is stored. At the detail level, the set of receipt lines that match to an invoice line is stored.

For each auto-match run, the following metrics about the run are stored:

- 1 The run date.
- 2 The number of invoices that matched exactly.
- 3 The number of invoices that matched within tolerance.
- 4 The total number of invoices processed.

Best terms calculations

Overview

The best terms calculation process compares the terms on the invoice and the terms on the PO, selects the most favorable term, and determines a terms date. The best terms calculation process is called after the auto-matching process, the online matching process, and the pre-paid invoice process.

After the best terms are calculated and the terms date is determined, the results are written to IM_DOC_HEAD for the invoice.

Terms ranking

Terms are ranked numerically. Terms with a lower ranking are preferable to terms with a higher ranking. During the best terms calculation, the ranks of the invoice terms and the PO terms are compared, and the terms with the lowest rank are selected as the best terms.

The terms ranking table (IM_TERMS_RANKING) stores the rankings. A flat file is used to populate the table with the terms ID and the terms ranking. The client is responsible for creating the flat file in the prescribed format. Terms ranking files are uploaded to ReIM on a periodic basis.

Supplier options

The following supplier options (IM_SUPPLIER_OPTIONS) affect the best terms calculation:

- Always Use Invoice Terms (USE_INVOICE_TERMS_IND)
When this indicator is set to 'Y', the supplier requires that invoice terms are always used; therefore, invoice terms and PO terms are not compared.
- ROG Date Allowed (ROG_DATE_ALLOWED_IND)
When this indicator is set to 'Y', the supplier allows the receipt of goods (ROG) date to be used to when determining the terms date. This indicator can only be set to 'Yes' if the Always Use Invoice Terms indicator is set to 'N'.

Terms date

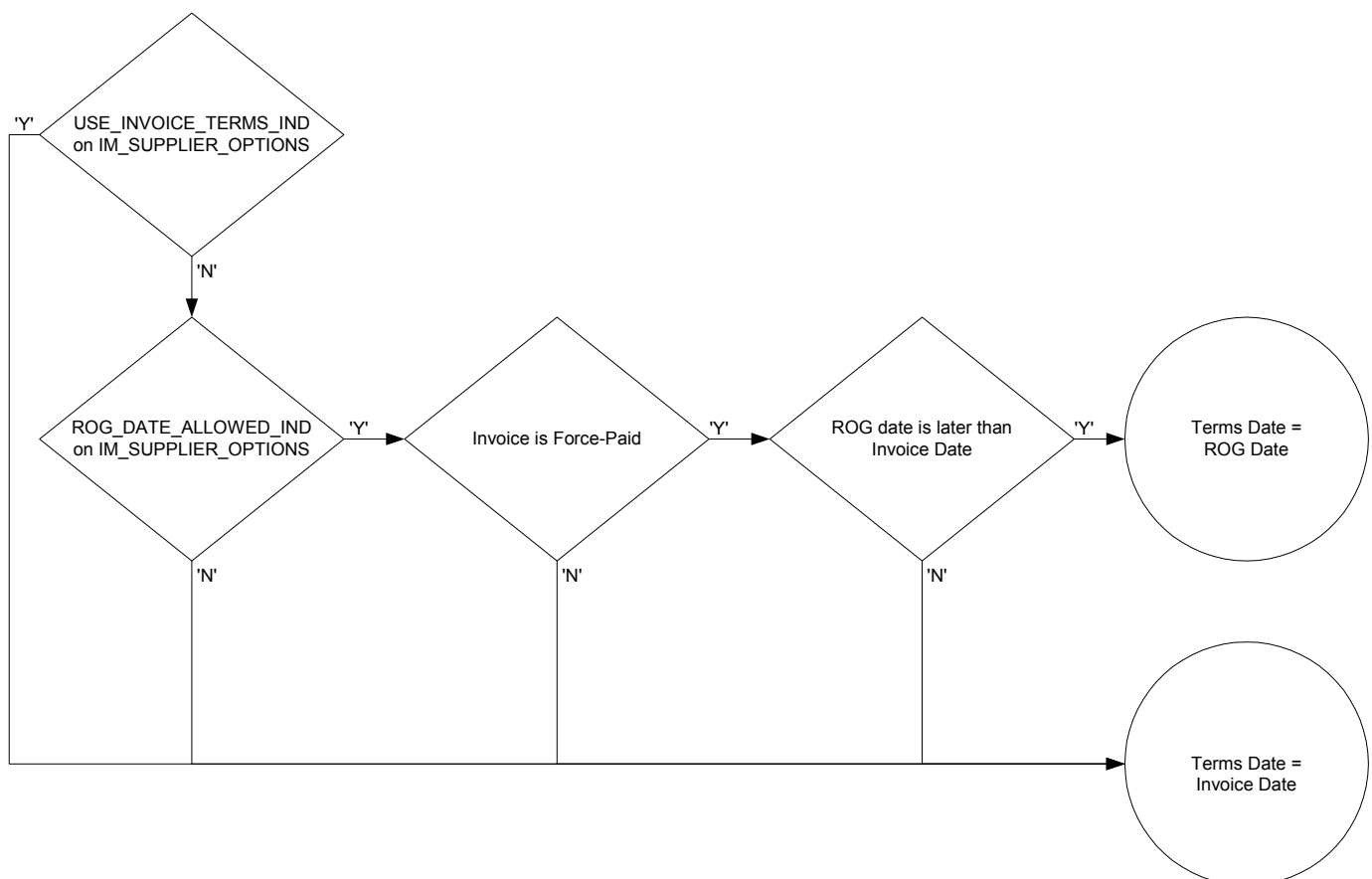
The terms date is either the invoice date or the receipt of goods (ROG) date. In most cases, the terms date is the invoice date. However, the ROG date is the terms date when all of the following is true:

- Always Use Invoice Terms (USE_INVOICE_TERMS_IND) on the supplier options table is set to 'N'.
- ROG Date Allowed (ROG_DATE_ALLOWED_IND) on the supplier options table is set to 'Y'.
- The invoice has been pre-paid.
- The ROG date is later than the invoice date.

Note: If there are multiple receipts for an invoice, the ROG date is the date of the last receipt.

How the terms date is determined

The following diagram describes how the terms date is determined:



The logic in the terms date determination

Terms-related data in the merchandising system (such as RMS)

The following terms data is stored in RMS:

- Terms description
- Discount days
- Percent

Assumptions and dependencies

- Best terms calculation applies only to merchandise invoices and to non-merchandise invoices.
- Merchandise invoices must have a status of 'MTCH' before the best terms calculation is performed.
- Non-merchandise invoices must have a status of 'APPRVE' before the best terms calculation is performed.
- When the supplier option 'Always Use Invoice Terms' is set to 'Y', the invoice terms are always used. The due date is calculated using the invoice date. The PO terms are never considered.
- The payment of invoices prior to or during matching does not update the matching status of the invoice. In these situations, a pre-paid invoice indicator is tripped to ensure the invoice is not paid a second time after matching and to trigger the correct accounting distribution. The best terms process is not re-invoked if the pre-paid indicator is set to 'Y'.
- The auto-matching process is run before the best terms calculation.

Primary tables involved

- IM_TERMS_RANKING
- IM_DOC_HEAD

Chapter 5 – Interfaces and file layouts

The EDI interface and layouts

Overview

Electronic Data Interchange (EDI) facilitates the computer-to-computer transmission of business information and transactions, such as invoices and purchase orders. EDI represents a convenient method by which a client and its suppliers can transfer information back and forth. The Voluntary Interindustry Commerce Standard (VICS) EDI is used by the general merchandise retail industry.

ReIM has two file-based EDI interfaces. Note that neither follows the VICS EDI standard. The ReIM EDI interfaces have been customized, and the client must translate them.

The interfaces represent the upload of invoices from a supplier and the download of documents to suppliers. These two common types of EDI are described below:

- EDI invoice upload is the standard description for an EDI process that uploads documents.
- EDI invoice download is the standard description for an EDI process that downloads Debit Memo, Credit Note Request, and Credit Memo data from ReIM to suppliers.

For information about ReIM's batch processes related to both of these types of EDI, see Chapter 7, "Batch processes".

Note that although the vast majority of invoices are created through either EDI upload or batch entry, users can also create invoices online and add details, or use the online dialog to add details to an invoice that was EDI uploaded.

The EDI reject table

The EDI invoice upload (ediupinv) batch process uploads invoices and credit notes from the EDI into the invoice-matching tables. This process validates the information in the file against itself and against the RMS (or equivalent merchandising system)/ReIM database. A limited set of data validation errors cause the invalid transaction to be written to error tables (IM_EDI_REJECT_DOC_xxx) where the data can be corrected through an online process.

The following errors are written to the EDI reject table for the user to manually correct through the front end:

- Supplier number (or Partner ID)
This value must be a valid supplier (SUPS table) or partner (PARTNER table) in RMS (or the equivalent merchandising system).

- Order number(s)
Order number(s) must be approved and created for the supplier or linked suppliers in RMS (or the equivalent merchandising system) on the ORDHEAD table. Non-merchandise invoices may not have any order numbers associated, so this validation should be skipped for this type of invoice.
- Order/location combination
The system validates that all order number/location combinations in the file are valid within RMS or the equivalent merchandising system (meaning that the relationship must exist on the ORDLOC table).
- Terms code
All terms must exist within RMS or the equivalent merchandising system on the TERMS table.
- Invoice date
A document cannot be older than the v-date minus the post-dated document days' system level parameter value or newer than the v-date.

Note: With regard to the interface with RMS 10.1, items must be transaction level. With regard to the interface with RMS 9, items must be a staple SKU, a fashion SKU, or a pack item (no fashion styles can be uploaded).

- Item number
Item numbers must exist within RMS on the ITEM_MASTER table (version 10.0) or the DESC_LOOK table (version 9). If a UPC or reference number is passed, this number should also be validated. The item number should also exist for the supplier.
- Duplicate invoice number for the supplier
The supplier's invoice number must be unique for the supplier.
- Merchandise invoices cannot be associated with a partner; they must only be associated with a supplier.
- Credit notes from a partner cannot have item records attached unless the partner type is a manufacturer, distributor, or wholesaler (type S1, S2, or S3).
- The system determines whether the invoice ID is valid if given.
- If the total quantity is given, the system determines whether the individual item quantities sum to total (the system only needs to check this if the supplier -level 'Match Total Qty' indicator is 'Yes').
- The system determines whether the total merchandise cost on the THEAD line matches the sum of costs from the TDETL lines (the sum of unit cost *qty).
- Either an item or a reference item must be specified on all documents except non-merchandise invoices or credit notes from a partner.
- The paid indicator must be either 'Y'es or 'N'o.

EDI invoice upload file layout

Input file format:

FHEAD	(1): Start of file.
THEAD	(1...n): Transaction (document) level info. Each file must have at least 1 THEAD.
TDETL	(0...n): Item detail records for this transaction. TDETL is optional for Credit Note docs.
TALLW	(0...n): Allowance records for this item. TALLW is optional.
TNMRC	(0...n): Non-merchandise records for this transaction. Required on non-merchandise documents, optional otherwise.
TTAIL	(1...n): Marks the end of a THEAD record. Each THEAD requires exactly one TTAIL.
FTAIL	(1): Marks the end of the file.

- TDETL and TNMRC do not need to occur in order. TALLW must follow TDETL
- If records are encountered in any order other than specified above, execution of process will halt.

- Example:

```
FHEAD
THEAD
TNMRC
FTAIL (no TTAIL encountered)
```

- If a record descriptor is encountered other than those specified in this document, execution of process will halt.
- Reject file will have an identical format. If no records are rejected, it will consist of only the FHEAD and FTAIL lines.
- All character variables should be right-padded with blanks and left justified; all numerical variables should be left-padded with zeroes and right-justified. Null variables should be blank.
- Single location invoices will be inserted into IM_DOC_HEAD, IM_INVOICE_DETAIL and IM_DOC_NON_MERCH. Multi-location invoices will be inserted into IM_PARENT_INVOICE, IM_PARENT_INVOICE_DETAIL and IM_PARENT_NON_MERCH.

- It is assumed all values that have dependent information included in the file (the location has dependent information of order NO, UPC, UPC-SUPP, and so on) will be valid for the RMS system. For example, the following is never anticipated to happen: only locations A, B and C exist in RMS; EDI reads a transaction that has location D. This sort of file may not be flagged as invalid in any way.
- All SKUs must have an associated primary UPC as the EDIUpload File only handles incoming UPCs.

Field Name	Field Type	Description	Req	Validation
FHEAD – File Header. First record of an upload file.				
Record descriptor	Char(5)	Describes file record type.	Y	Halt execution if not FHEAD.
Line id	Number(10)	Sequential file line number.	Y	Halt execution if not 0000000001.
Gentran ID	Char(5)	The type of transaction this file represents.	Y	Halt execution if not UPINV.
Current date	Char(14)	File date in YYYYMMDDHH24MISS format.	Y	Halt execution if invalid date format.
THEAD – Transaction Header. Start of a document transaction.				
Record descriptor	Char(5)	Describes file record type.	Y	THEAD
Line id	Number (10)	Sequential file line number.	Y	Halt execution if not in sequence.
Transaction number	Number(10)	Sequential transaction number. All records within this transaction will also have this transaction number.	Y	Reject entire file if: transaction number is not numeric or not in sequence first transaction number is not 0000000001

Field Name	Field Type	Description	Req	Validation
Document Type	Char(6)	<p>Describes the type of document being uploaded. The document type will determine the types of detail information that are valid for the document upload. Stored in IM_DOC_HEAD.TYPE.</p> <p>Valid values are:</p> <p>MRCHI – Merchandise Invoice</p> <p>NMRCHI – Non-Merchandise Invoice</p> <p>CRDNT – Credit Note</p>	Y	<p>Reject transaction to file if</p> <p>document type is null</p> <p>document type is not MRCHI (merchandise invoice), NMRCHI (non-merchandise invoice), or CRDNT (credit note)</p> <p>document type is CRDNT (credit note); vendor is not a supplier, manufacturer, distributor, or wholesaler.</p> <p>document type is CRDNT and TALLW records exist</p> <p>document type is MRCHI and item detail records DO exist for this transaction (this type of transaction must have no item detail records)</p> <p>location or location type are null</p> <p>document type is CRDNT or NMRCHI and <i>any</i> error occurs with the document</p>

Field Name	Field Type	Description	Req	Validation
Vendor Document Number	Char (30)	Vendor's document number. Stored in IM_DOC_HEAD.EXT_DOC_ID with all characters converted to their upper case (i.e., ThisDocId -> THISDOCID).	Y	<p>Reject entire upload file if:</p> <p>The same vendor document number occurs more than once in the file.</p> <p>Reject transaction to file if:</p> <p>vendor document number is null</p> <p>vendor document number is not unique for this vendor.</p> <p>vendor document number is not alpha-numeric</p>
Vendor Type	Char(6)	<p>Type of vendor (either supplier or partner) for this document. Stored in IM_DOC_HEAD.VENDOR_TYPE</p> <p>Valid values are:</p> <p>SUPP – Supplier</p> <p>BK - Bank</p> <p>AG - Agent</p> <p>FF – Freight Forwarder</p> <p>IM - Importer</p> <p>BR - Broker</p> <p>FA - Factory</p> <p>AP - Applicant</p> <p>CO - Consolidator</p> <p>CN - Consignee</p> <p>S1 – Merch Supp level 1</p> <p>S2 – Merch Supp level 2</p> <p>S3 – Merch Supp level 3</p>	Y	<p>Reject transaction to file if:</p> <p>vendor type is null or if it is not a valid vendor type (from Vendor class)</p> <p>document type is MRCHI (merchandise invoice) and vendor type is not 'S'upplier</p>

Field Name	Field Type	Description	Req	Validation
Vendor ID	Char(10)	Vendor for this document. Stored in IM_DOC_HEAD.VENDOR _TYPE	Y	<p>Reject transaction to file if: vendor ID is null vendor type is partner and vendor ID is not valid partner</p> <p>Reject transaction to tables if: vendor is a supplier and supplier is not valid vendor is a supplier and vendor ID is not completely numeric</p>
Vendor Document Date	Char(14)	Date document was issued by the vendor (in YYYYMMDDHH24MISS format). Stored in IM_DOC_HEAD.DOC_DATE	Y	<p>Reject transaction to file if: vendor document date is null date is not a valid date format</p> <p>Reject transaction to tables if: vendor document date is after the vdate or before (vdate – post_dated_doc_days) (from im_system_options)</p>

Field Name	Field Type	Description	Req	Validation
Order Number	Number(8)	Merchandising system order number for this document. Required for merchandise invoices and optional for others. Store in IM_DOC_HEAD.ORDER_NO.	N	<p>Reject transaction to file if:</p> <ul style="list-style-type: none"> order number exists and is not numeric order number is null and vendor type is a supplier order number exists and vendor type is <i>not</i> a supplier order number exists and location or location type are null <p>Reject transaction to tables if:</p> <ul style="list-style-type: none"> order number exists but is not valid for the supplier or the supplier's linked suppliers order number exists but is not valid for the location/location type
Location	Number(10)	Merchandising system location for this document. Required for merchandise invoices and optional for others. Stored in IM_DOC_HEAD.LOCATION.	Y	<p>Reject transaction to file if:</p> <ul style="list-style-type: none"> location or location type do not exist location exists and is not numeric location exists and location type is not 'S'tore or 'W'arehouse <p>Reject transaction to tables if:</p> <ul style="list-style-type: none"> location and location type exist but are not valid

Field Name	Field Type	Description	Req	Validation
Location Type	Char(1)	Merchandising system location type (either 'S'tore or 'W'arehouse) for this document. Required for merchandise invoices and optional for others. Stored in IM_DOC_HEAD.LOC_TYPE.	N	Reject transaction to file if: location type exists and location is null
Terms	Char(15)	Terms of this document. If terms are not provided, the vendor's default terms will be associated with this record. Stored in IM_DOC_HEAD.TERMS. This value is used to get the Terms Discount Percentage to be stored on IM_DOC_HEAD.TERMS_DSCNT_PCT.	N	Reject transaction to tables if: terms exist and are not valid
Due Date	Char(14)	Date the amount due is due to the vendor (YYYYMMDDHH24MISS format). If due date is not provided, default due date is calculated based on vendor and terms. Stored in IM_DOC_HEAD.DUE_DATE.	N	Reject transaction to file if: due date exists and is not a valid date format due date is before the vendor document date
Payment method	Char(6)	Method for paying this document. Stored in IM_DOC_HEAD.PAYMENT_METHOD.	N	Reject transaction to file if: payment method exists and is not valid
Currency code	Char(3)	Currency code for all monetary amounts on this document. Stored in IM_DOC_HEAD.CURRENCY_CODE.	Y	Reject transaction to file if: currency code is null currency code is not valid order number exists and currency code does not match the order's currency

Field Name	Field Type	Description	Req	Validation
Exchange rate	Number(12,4)	Exchange rate for conversion of document currency to the primary currency. Stored in IM_DOC_HEAD.EXCHANGE_RATE.	N	Reject transaction to file if: exchange rate exists and is not numeric
Sign Indicator	Char(1)	Indicates either a positive (+) or a negative (-) total cost amount.	Y	Reject transaction to file if sign indicator is null or if it is not '+' or '-'.
Total Cost	Number(20,4)	Total document cost, including all items and costs on this document. This value is in the document currency. Stored in IM_DOC_HEAD.TOTAL_COST and IM_DOC_HEAD.RESOLUTION_ADJUSTED_TOTAL_COST.	Y	Reject transaction to file if: total cost is null total cost is not numeric total cost does not equal the sum of extended costs for all item detail records in this transaction total cost is not negative and vendor document type is CRDNT
Sign Indicator	Char(1)	Indicates either a positive (+) or a negative (-) total quantity amount.	Y	Reject transaction to file if sign indicator is null or if it is not '+' or '-'.
Total Quantity	Number(12,4)	Total quantity of items on this document. This value is in EACHES (no other units of measure are supported in ReIM). Stored in IM_DOC_HEAD.TOTAL_QTY and IM_DOC_HEAD.RESOLUTION_ADJUSTED_TOTAL_QTY.	Y	Reject transaction to file if: total quantity is null total quantity is not numeric total quantity does not equal the sum of quantities for all item detail records in this transaction total quantity is not zero when vendor document type is 'NMRCHI'

Field Name	Field Type	Description	Req	Validation
Sign Indicator	Char(1)	Indicates either a positive (+) or a negative (-) total discount amount.	Y	Reject transaction to file if sign indicator is null or if it is not '+' or '-'.
Total Discount	Number(12,4)	Total discount applied to this document. This value is in the document currency. Stored in IM_DOC_HEAD.TOTAL_DISCOUNT	Y	Reject transaction to file if: total discount is null total discount is not numeric
Freight Type	Char(6)	The freight method for this document.	N	Reject transaction to file if: freight type exists and is not valid
Paid Ind	Char(1)	Indicates if this document has been paid. Stored in IM_DOC_HEAD.MANUALLY_PAID_IND.	Y	Reject transaction to file if: paid ind is null paid ind is not Y or N
Multi-Location	Char(1)	Indicates if this invoice goes to multiple location. If Yes, the record should be inserted to IM_PARENT_INVOICE table.	Y	Reject transaction to file if: multi-location is null multi-location is not Y or N
Custom Document Reference 1	Char(30)	This optional field is included in the upload file for client customization. No validation will be performed on this field. Stored in IM_DOC_HEAD.CUSTOM_REF_1.	N	
Custom Document Reference 2	Char(30)	This optional field is included in the upload file for client customization. No validation will be performed on this field. Stored in IM_DOC_HEAD.CUSTOM_REF_2.	N	
Custom Document Reference 3	Char(30)	This optional field is included in the upload file for client customization. No validation will be performed on this field. Stored in IM_DOC_HEAD.CUSTOM_REF_3.	N	

Field Name	Field Type	Description	Req	Validation
Custom Document Reference 4	Char(30)	This optional field is included in the upload file for client customization. No validation will be performed on this field. Stored in IM_DOC_HEAD.CUSTOM_REF_4.	N	
Cross-reference document number	Number(10)	Document that a credit note is for. Blank for all document types other than merchandise invoices. Stored in IM_DOC_HEAD.REF_DOC.	N	Reject transaction to file if: cross-reference document number exists and is not numeric
TDETL – Item Detail Record. This information will be inserted into the IM_INVOICE_DETAIL table for Merchandise Invoice and IM_DOC_DETAIL_REASON_CODES for Credit Notes.				
Record descriptor	Char(5)	Describes file record type.	Y	TDETL
Line id	Number(10)	Sequential file line number.	Y	Halt execution if not in sequence.
Transaction number	Number(10)	Transaction number for this item detail record.	Y	Reject entire file if: transaction number is not numeric transaction number is not the same as the current transaction
UPC	Char(25)	UPC for this detail record. Valid item number will be retrieved for the UPC. Stored in IM_INVOICE_DETAIL.ITEM or IM_DOC_DETAIL_REASON_CODES.ITEM.	Y	Reject transaction to file if: UPC is null Reject transaction to tables if: Valid item is not found for UPC and UPC supp Valid item is not associated with the supplier The item found is identical to another detail item for this transaction (no duplicate items).

Field Name	Field Type	Description	Req	Validation
UPC Supplement	Number(5)	Supplement for the UPC. Note: UPC Supp is only valid for RMS 9 implementation. For RMS 10.1 implementation, this field will <i>always</i> be blank.	N	Reject transaction to file if: UPC supplement exists and is not numeric
Sign Indicator	Char(1)	Indicates either a positive (+) or a negative (-) Original Document Quantity amount.	Y	Reject transaction to file if sign indicator is null or if it is not '+' or '-'.
Original Document Quantity	Number(12,4)	Quantity, in EACHES, of the item on this detail record. Stored in IM_INVOICE_DETAIL.INVOICE_QTY and IM_INVOICE_DETAIL.RESOLUTION_ADJUSTED_QTY.	Y	Reject transaction to file if: original document quantity is null original document quantity is not numeric
Sign Indicator	Char(1)	Indicates either a positive (+) or a negative (-) Original Unit Cost amount.	Y	Reject transaction to file if sign indicator is null or if it is not '+' or '-'.
Original Unit cost	Number(20,4)	Unit cost, in document currency, of the item on this detail record. Stored in IM_INVOICE_DETAIL.UNIT_COST and IM_INVOICE_DETAIL.RESOLUTION_ADJUSTED_UNIT_COST.	Y	Reject transaction to file if: original unit cost is null original unit cost is not numeric
Sign Indicator	Char(1)	Indicates either a positive (+) or a negative (-) total allowance. Default is "+" if no allowances exist for this detail record.	Y	Reject transaction to file if: sign indicator is null sign indicator is not '+' or '-'

Field Name	Field Type	Description	Req	Validation
Total Allowance	Number(20,4)	Sum of allowance details for this item detail record. If no allowances exist for this item detail record, value will be 0.	Y	Reject transaction to file if: total allowance is null total allowance is not numeric total allowance does not equal the sum of allowance amounts for all allowance records in this item detail record total allowance is not 0 and vendor document type is CRDNT
TALLW – Allowance Record. This information will be inserted into IM_INVOICE_DETAIL_ALLOWANCE table.				
Record descriptor	Char(5)	Describes file record type.	Y	TALLW
Line id	Number(10)	Sequential file line number.	Y	Halt execution if not in sequence.
Transaction Number	Number(10)	Transaction number for this item allowance record.	Y	Reject entire file if: transaction number is not numeric transaction number is not the same as the current transaction
Allowance code	Char(14)	Allowance code for this allowance record. Stored in IM_INVOICE_DETAIL_ALLOWANCE.ALLOWANCE_CODE.	Y	Reject transaction to file if: allowance code is null allowance code is not valid
Sign Indicator	Char(1)	Indicates either a positive (+) or a negative (-) allowance amount.	Y	Reject transaction to file if sign indicator is null or if it is not '+' or '-'.
Allowance Amount	Number (20,4)	Amount of allowance in document currency. Stored in IM_INVOICE_DETAIL_ALLOWANCE.ALLOWANCE_AMOUNT.	Y	Reject transaction to file if allowance amount is null or not numeric.

Field Name	Field Type	Description	Req	Validation
TNMRC – Non-Merchandise Record. Records of this type will contain non-merchandise costs. These costs will be inserted into the IM_DOC_NON_MERCH table. Non-merchandise costs records are only required when the document type is non-merchandise. Non-merchandise cost records can also be associated with merchandise type documents if the vendor associated with the document allows non-merch costs on merchandise invoices (IM_SUPPLIER_OPTIONS. MIX_MERCH_NON_MERCH_IND).				
Record descriptor	Char(5)	Describes file record type.	Y	TNMRC
Line id	Number (10)	Sequential file line number.	Y	Halt execution if not in sequence.
Transaction number	Number(10)	Transaction number for this non-merchandise record.	Y	Reject entire file if: transaction number is not numeric transaction number is not the same as the current transaction
Non Merchandise Code	Char(6)	Non-Merchandise code that describes this cost. Stored in IM_DOC_NON_MERCH.NON_MERCH_CODE.	Y	Reject transaction to file if: non-merchandise code is null non-merchandise code is not valid
Sign Indicator	Char(1)	Indicates either a positive (+) or a negative (-) Non-Merchandise Amt.	Y	Reject transaction to file if sign indicator is null or if it is not '+' or '-'.
Non Merchandise Amt	Number(20,4)	Cost in the document currency. Stored in IM_DOC_NON_MERCH.NON_MERCH_AMT.	Y	Reject transaction to file if: non-merchandise amount is null non-merchandise amount is not numeric. non-merchandise amount does not have a negative value and this is part of a credit note document (THEAD.Vendor Document Type = 'CRDNT').

Field Name	Field Type	Description	Req	Validation
Service Performed Indicator	Char(1)	Indicates if a service has actually been performed. Stored in IM_DOC_NON_MERCH.SERVICE_PERF_IND.	Y	Reject transaction to file if: service performed indicator is null service performed indicator is not Y or N
Store	Number(10)	Store at which the service was performed. Stored in IM_DOC_NON_MERCH.STORE.	N	Reject transaction to file if: store exists and is not numeric service performed indicator is Y and store is not valid
TTAIL – Transaction Tail. Marks the end of a transaction.				
Record descriptor	Char(5)	Describes file record type.	Y	TTAIL
Line id	Number(10)	Sequential file line number.	Y	Halt execution if not in sequence.
Transaction number	Number(10)	Transaction number for the transaction that this record is closing.	Y	Reject entire file if: transaction number is not numeric transaction number is not the same as the current transaction.
Transaction lines	Number(6)	Total number of detail lines within this transaction.	Y	Reject transaction to file if transaction lines is not numeric, if it does not match the count of lines within the transaction, or if it is zero (transaction must have details).
FTAIL – File TAIL. Marks the end of the upload file.				
Record descriptor	Char(5)	Describes file record type.	Y	FTAIL
Line id	Number(10)	Sequential file line number.	Y	Halt execution if not in sequence.

Field Name	Field Type	Description	Req	Validation
Number of lines	Number(10)	Total number of lines within this file not counting FHEAD and FTAIL.	Y	Halt execution if number of lines is not numeric, if it does not match the count of lines within the file (excluding FHEAD and FTAIL), or if it is 2 (FHEAD and FTAIL only, file has no transactions).

Additional notes

- Many fields on the database are not uploaded as part of this file. The table below lists all fields on the tables and the information source.

Database Table	Database Column	Data Source
IM_DOC_HEAD	DOC_ID	Generated by nextval from sequence IM_DOC_HEAD_SEQ
	TYPE	Uploaded from file - THEAD Document Type
	STATUS	Defaulted to 'RMTCH' – ready for match.
	ORDER_NO	Uploaded from file – THEAD Order No
	LOCATION	Uploaded from file – THEAD Location
	LOC_TYPE	Uploaded from file – THEAD Location Type
	TOTAL_DISCOUNT	Uploaded from file – THEAD Total Discount
	GROUP_ID	Defaulted NULL – group id is only set for manually entered documents.
	PARENT_ID	Defaulted NULL – parent id is only set after multi-loc invoices are split.
	DOC_DATE	Uploaded from file – THEAD Vendor Document Date
	CREATE_DATE	Defaulted to current date/time.
	CREATE_ID	Defaulted to EDI_UPLOAD
	VENDOR_TYPE	Uploaded from file – THEAD Vendor Type
	VENDOR	Uploaded from file – THEAD Vendor
	EXT_DOC_ID	Uploaded from file – THEAD Vendor Document Number
	EDI_UPLOAD_IND	Defaulted to 'Y' because record is uploaded through EDI.

Database Table	Database Column	Data Source
	EDI_DOWNLOAD_IND	Defaulted to 'N' because record has just been uploaded and therefore has not been downloaded to any financials systems.
	TERMS	Uploaded from file – THEAD Terms.
	TERMS_DSCNT_PCT	Uploaded from file – THEAD Terms Discount Percentage.
	DUE_DATE	Uploaded from file – THEAD Due Date
	PAYMENT_METHOD	Uploaded from file – THEAD Payment Method
	MATCH_ID	Defaulted to NULL because the document is just being inserted to the system and therefore is not matched.
	MATCH_DATE	Defaulted to NULL because the document is just being inserted to the system and therefore is not matched.
	APPROVAL_ID	Defaulted to NULL because the document is just being inserted to the system and therefore is approved.
	APPROVAL_DATE	Defaulted to NULL because the document is just being inserted to the system and therefore is not matched.
	PRE_PAID_IND	Defaulted to 'N' because the document is just being inserted to the system and therefore has not been forced paid.
	PRE_PAID_ID	Defaulted to NULL because the document is just being inserted to the system and therefore has not been forced paid.
	POST_DATE	Defaulted to NULL because the document is just being uploaded and therefore has not been posted to a financial system.
	CURRENCY_CODE	Uploaded from file – THEAD Currency Code.
	EXCHANGE_RATE	Uploaded from file – THEAD Exchange Rate.
	TOTAL_COST	Uploaded from file – THEAD Control Total Cost.
	TOTAL_QTY	Uploaded from file – THEAD Control Total Cost.
	MANUALLY_PAID_IND	Uploaded from file – THEAD Paid Ind.

Database Table	Database Column	Data Source
	PAYMENT_REF_NO	Uploaded from file – THEAD Payment Reference Number.
	PAYMENT_DATE	Uploaded from file – THEAD Payment Reference Date.
	CUSTOM_DOC_REF_1	Uploaded from file – THEAD Custom Document Reference 1.
	CUSTOM_DOC_REF_2	Uploaded from file – THEAD Custom Document Reference 2.
	CUSTOM_DOC_REF_3	Uploaded from file – THEAD Custom Document Reference 3.
	CUSTOM_DOC_REF_4	Uploaded from file – THEAD Custom Document Reference 4.
	LAST_UPDATE_ID	Defaulted to 'EDI_UPLOAD' because the batch process has just inserted the record.
	LAST_UPDATE_DATE TIME	Defaulted to the current date/time because the batch process has just inserted the record.
	FREIGHT_TYPE	Uploaded from file – THEAD Freight Type.
	REF_DOC	Uploaded from file – THEAD Cross-reference document number
	REF_AUTH_NO	Defaulted to NULL
	COST_PRE_MATCH	Defaulted to 'N'
	DETAIL_MATCHED	Defaulted to 'N'
	BEST_TERMS	Defaulted to NULL
	BEST_TERMS_SOURCE	Defaulted to NULL
	VARIANCE	Defaulted to NULL
IM_DOC_NON_MERCH	DOC_ID	Inserted by batch process with value for the transaction generated for the IM_DOC_HEAD.DOC_ID.
	NON_MERCH_CODE	Uploaded from file – TNMRC Non-Merchandise Cost.
	NON_MERCH_AMT	Uploaded from file – TNMRC Non-Merchandise Amount.
	SERVICE_PERF_IND	Uploaded from file – TNMRC Service Performed Indicator
	STORE	Uploaded from file – TNMRC Store

Database Table	Database Column	Data Source
IM_INVOICE_DETAIL	DOC_ID	Inserted by batch process with value for the transaction generated for the IM_DOC_HEAD.DOC_ID.
	ITEM	Uploaded from file – TDETL Item. If TDETL Item is null (and only the REF_ITEM is in the file), the process will look up the ITEM associated with the REF_ITEM and insert the ITEM value.
	UNIT_COST	Uploaded from file – TDETL Original Document Cost
	INVOICE_QTY	Uploaded from file – TDETL Original Document Quantity
	STATUS	Defaults to UNMTCH – Unmatched because the document item is just being uploaded and therefore is unmatched.
	RESOLUTION_ADJUSTED_UNIT_COST	Initialized to be the same as the UNIT_COST.
	RESOLUTION_ADJUSTED_QTY	Initialized to be the same as the INVOICE_QTY.
	COST_MATCHED	Defaults to N because the document item is just being uploaded and therefore has not been matched and flagged with a cost discrepancy.
	QTY_MATCHED	Defaults to N because the document item is just being uploaded and therefore has not been matched and flagged with a quantity discrepancy.
	ADJUSTMENT_PENDING	Defaulted to N because the document item is just being uploaded and therefore has not been matched and flagged with pending receiver adjustments.
	LAST_UPDATE_ID	Defaulted to 'EDI_UPLOAD' because the batch process is just inserting the record.
	LAST_UPDATE_DATETIME	Defaulted to 'EDI_UPLOAD' because the batch process is just inserting the record.
IM_DOC_DETAIL_REASON_CODES	DOC_ID	Inserted by batch process with value for the transaction generated for the IM_DOC_HEAD.DOC_ID.

Database Table	Database Column	Data Source
	ITEM	Uploaded from file – TDETL Item. If TDETL Item is null (and only the REF_ITEM is in the file), the process will look up the ITEM associated with the REF_ITEM and insert the ITEM value.
	REASON_CODE_ID	Null for Credit Notes
	STATUS	Defaults to APPRVE – Approved because the document item is just being uploaded and therefore is in approved status.
	ADJUSTED_UNIT_COST	Uploaded from file – TDETL Original Document Unit Cost
	ADJUSTED_QTY	Uploaded from file – TDETL Original Document Item Quantity
	LAST_UPDATE_ID	Defaulted to 'EDI_UPLOAD' because the batch process is just inserting the record.
	LAST_UPDATE_DATE TIME	Defaulted to 'EDI_UPLOAD' because the batch process is just inserting the record.
IM_INVOICE_ DETAIL_ ALLOWANCE	DOC_ID	Inserted by batch process with value for the transaction generated for the IM_DOC_HEAD.DOC_ID.
	ITEM	Same item that is inserted on IM_INVOICE_DETAIL.
	ALLOWANCE_CODE	Uploaded from file – TALLW allowance code
	ALLOWANCE_ AMOUNT	Uploaded from file – TALLW allowance amount (positive or negative depending on TALLW sign indicator)

EDI invoice download file layout

Output file format:

FHEAD	(1): Start of file.
THEAD	(1...n): Transaction (document) level info. Each file must have at least 1 THEAD.
TDETL	(0...n): Item detail records for this transaction. TDETL is always required.
TNMRC	(0...n): Non-merchandise records for this transaction. Required on non-merchandise documents, optional otherwise.
TTAIL	(1...n): Marks the end of a THEAD record. Each THEAD requires exactly one TTAIL.
FTAIL	(1): Marks the end of the file.

- If records are encountered in any order other than specified above, execution of process will halt.

Example:

FHEAD

THEAD

TNMRC

FTAIL (no TTAIL encountered)

- If a record descriptor is encountered other than those specified in this document, execution of process will halt.
- All character variables should be right-padded with blanks and left justified; all numerical variables should be left-padded with zeroes and right-justified. Null variables should be blank.

Note: The file will *not* be threaded, but rather ordered by vendor id (THEAD). It is assumed that this file will be broken out by vendor id during the translation process.

Field Name	Field Type	Description	Req	Validation
FHEAD – File Header. First record of an upload file.				
Record descriptor	Char(5)	FHEAD	Y	
Line id	Number(10)	Generated Sequential file line number.	Y	
Gentran ID	Char(5)	DNINV	Y	

Field Name	Field Type	Description	Req	Validation
Current date	Char(14)	File date in YYYYMMDDHH24MISS format.	Y	
THEAD – Transaction Header. Start of a document transaction.				
Record descriptor	Char(5)	THEAD	Y	
Line id	Number (10)	Generated Sequential file line number.	Y	
Transaction number	Number(10)	Sequential transaction number. All records within this transaction will also have this transaction number.	Y	
Document Type	Char(6)	Describes the type of document being downloaded. The document type will determine the types of detail information that are valid for the document downloaded. Retrieved from IM_DOC_HEAD.TYPE where type is debit memo, credit note request or credit memo and in Approved or Posted Status.	Y	Debit memo, credit note request cost, credit note request quantity, credit memos in approved status
Vendor Document Number	Char (30)	Vendor's document number. Retrieved from IM_DOC_HEAD.EXT_DOC_ID.	Y	
Invoice Number	Char(6)	Corresponding invoice resolved by the document. Retrieved from IM_DOC_HEAD.REF_DOC.	Y	
Vendor ID	Number(10)	Vendor for this document. Retrieved from IM_DOC_HEAD.VENDOR	Y	
Document Date	Char(14)	Date the document was entered into the system in YYYYMMDDHH24MISS format. Retrieved from IM_DOC_HEAD.DOC_DATE	Y	

Field Name	Field Type	Description	Req	Validation
Order	Number(8)	Order number for this document, if any. Retrieved from IM_DOC_HEAD.ORDER_NO	N	
Location	Number(10)	Location for this document, if any. Retrieved from IM_DOC_HEAD.LOCATION.	N	
Location Type	Char(1)	Location type for this document, if any. Retrieved from IM_DOC_HEAD.LOC_TYPE.	N	
Terms	Char(15)	Terms of this document. Retrieved from IM_DOC_HEAD.TERMS.	N	
Due Date	Char(14)	Date the amount due is due from the vendor (YYYYMMDDHH24MISS format). Retrieved from IM_DOC_HEAD.DUE_DATE.	N	
Currency Code	Char(3)	Currency code for this document. Retrieved from IM_DOC_HEAD.CURRENCY_CODE.		
Exchange rate	Number(12,4)	Exchange rate for conversion of document currency to the primary currency. Retrieved from IM_DOC_HEAD.EXCHANGE_RATE.	N	
Sign indicator	Char(1)	Indicates either a positive (+) or a negative (-) total cost.	Y	
Total Cost	Number(20,4)	Total document cost, including all items and costs on this document. This value is in the document currency. Retrieved from IM_DOC_HEAD.TOTAL_COST.	Y	

Field Name	Field Type	Description	Req	Validation
Sign indicator	Char(1)	Indicates a positive (+) quantity.	Y	
Total Quantity	Number(12,4)	Total quantity of items on this document. This value is in EACHES (no other units of measure are supported in ReIM). Retrieved from IM_DOC_HEAD.TOTAL_QTY.	Y	
TDETL – Item Detail Record. This information will be inserted into the IM_DOC_DETAIL_REASON_CODES table.				
Record descriptor	Char(5)	TDETL	Y	
Line id	Number(10)	Generated Sequential file line number.	Y	
Transaction number	Number(10)	Generated Transaction number for this item detail record.	Y	
SKU	Char(25)	Internal SKU/Item for this document. This is always sent. Retrieved from IM_DOC_DETAIL.ITEM NOTE: UPC is used for RMS 9 and Item is used for RMS 10.1.	Y	
UPC	Char(25)	UPC for this detail record. Retrieved from UPC_EAN.UPC (RMS 9) or ITEM_MASTER.ITEM (RMS 10.1). This field is sent if available. Note: UPC is used for RMS 9 and Ref-Item is used for RMS 10.1. Ref-Item consists of UPC and UPC-Supp appended together with a separating hyphen(-).	N	

Field Name	Field Type	Description	Req	Validation
UPC Supplement	Number(5)	Supplement for the UPC. Retrieved from UPC_EAN.UPC_SUPPLEMENTS. This field is sent if available. Note: UPC Supp is only valid for 9 implementation. For 10.1 implementation, this field will <i>always</i> be blank.	N	
VPN	Char(30)	Vendor Product Number. This field is sent if available. Retrieved from ITEM_SUPPLIER.VPN.	N	
Comments	Char(200)	Comments associated with Reason Code. Retrieved from IM_DOC_DETAIL_COMMENTS.TEXT	Y	
Reason Code	Char(6)	Reason Code for this document. Retrieved from IM_DOC_DETAIL_REASON_CODES.REASON_CODE_ID	Y	
Reason Code description	Char(50)	Description associated with Reason Code. Retrieved from IM_REASON_CODES.REASON_CODE_DESC		
Sign indicator	Char(1)	Indicates a positive (+) discrepant qty.	Y	
Discrepant Quantity	Number(12,4)	Quantity, in EACHES, of the item that is discrepant for this detail record. Retrieved from IM_DOC_DETAIL_REASON_CODES.ADJUSTED_QTY.	Y	
Sign Indicator	Char(1)	Indicates either a positive (+) or a negative (-) discrepant cost.	Y	
Discrepant cost	Number(20,4)	Unit cost, in document currency, of the item that is discrepant for this detail record. Retrieved from IM_DOC_DETAIL_REASON_CODES.ADJUSTED_UNIT_COST.	Y	

Field Name	Field Type	Description	Req	Validation
TNMRC – Non-Merchandise Record. Records of this type will contain non-merchandise costs. These costs will be retrieved from the IM_DOC_NON_MERCH table. Non-merchandise cost records are only required when the document type is non-merchandise. Non-merchandise cost records can also be associated with merchandise type documents if the vendor associated with the document allows non-merch costs on merchandise invoices (IM_SUPPLIER_OPTIONS. MIX_MERCH_NON_MERCH_IND).				
Record descriptor	Char(5)	TNMRC	Y	
Line id	Number (10)	Generated Sequential file line number.	Y	
Transaction number	Number(10)	Generated Transaction number for this non-merchandise record.	Y	
Non Merchandise Code	Char(6)	Non-Merchandise code that describes this cost. Retrieved from IM_DOC_NON_MERCH.NON_MERCH_CODE.	Y	
Sign indicator	Char(1)	Indicates either a positive (+) or a negative (-) non merchandise amount.	Y	
Non Merchandise Amt	Number(20,4)	Cost in the document currency. Retrieved from IM_DOC_NON_MERCH.NON_MERCH_AMT.	Y	
TTAIL – Transaction Tail. Marks the end of a transaction.				
Record descriptor	Char(5)	TTAIL	Y	
Line id	Number(10)	Generated Sequential file line number.	Y	
Transaction number	Number(10)	Generated Transaction number for the transaction that this record is closing.	Y	
Transaction lines	Number(6)	Total number of detail lines within this transaction.	Y	
FTAIL – File TAIL. Marks the end of the upload file.				
Record descriptor	Char(5)	FTAIL.	Y	
Line id	Number(10)	Generated Sequential file line number.	Y	

Field Name	Field Type	Description	Req	Validation
Number of lines	Number(10)	Total number of lines within this file not counting FHEAD and FTAIL.	Y	

The merchandise system interface

The client's merchandising system provides basic data about the purchase orders and receipts to which ReIM matches invoices. ReIM accesses this reference information for invoices (for example, supplier details, purchase order information, receipt information, and so on) using an interface data access layer (DAL) to the merchandising system. The DAL is a series of isolated SQL statements that involve merchandising system tables.

ReIM holds invoice matching-specific information in its own tables. The vast majority of the DAL reads information from the merchandising system. A small portion of the DAL writes the results of invoice matching to the merchandising system's record for the receipt.

ReIM will be delivered with DALs for both base RMS 9 and base RMS 10.1.

ReIM has been designed to work with any merchandising system. For more information about the design of ReIM's DAL layer, see Chapter 3, "Technical architecture".

A word about indexes and RMS 9

Some indexes in the DDL should only be included when the client is running ReIM against RMS 9. The index names are the following:

- IM_PARTIALLY_MATCHED_REC_I1
- IM_QTY_DISCREPANCY_I2

The reason for this inclusion is that SKU is a number(8) in rms9. In rms10, however, item is a varchar2. Numbers can always go into varchar fields, thus, ReIM tables define an item as a varchar2(25). In places within the auto-match process, ReIM (when running against rms9) must be able to join between shipsku.sku and im_partially_matched_receipts.item (in another place, ReIM has a join between shipsku.sku and im_qty_discrepancy.item, and ReIM has two special indexes that only belong in 9). This processing compares a number to a string. Oracle can perform the same processing implicitly, but slowly. Instead of letting Oracle perform this processing, ReIM has a function based index on its tables that converts the varchar value to a number. In essence, the process is comparing a number to a number (much faster). The problem is that these indexes are only valid on the ReIM tables when ReIM is running against RMS 9. In RMS 10.1, items are characters. If ReIM attempted to convert the item '122003934-1234' to a number, problems would arise.

Interface dataflows

To understand the type of data that is exchanged between ReIM and the merchandising system, see Chapter 4, "Functional design".

Porting

If a client wishes to implement ReIM with another merchandising system (or a different or with custom version of RMS), the client must port the interface DAL so that ReIM has access to the correct merchandising system reference information. Porting the interface DAL is a straightforward process.

The interface DAL code is segmented from the rest of the application code. The main task in porting the interface DAL is in the changing of the SQL code to reference the tables in the applicable merchandising system. The primary skill needed for the porting process is a knowledge of the merchandising system's schema and some basic SQL. Java knowledge is needed to compile, test and deploy the application using the new interface DAL.

DAL beans

Each element of the ReIM interface DAL has two components: a general, abstract bean that defines the bean signature and a concrete bean implementation that connects to the database. ReIM provides concrete beans that reference the following merchandising systems: base RMS 9 and base RMS 10.1.

At runtime, a bean factory controller determines which concrete DAL implementation to use by consulting the properties file, `com.retek.reim.reim.properties`.

The following tables illustrate the ReIM DAL to RMS 9 and to RMS 10.1 interfaces:

RMS 9 referencing beans and APIs	
AddrBean	ClassBean
CodeDetailBean	CurrenciesBean
CurrencyConversionAPI	CurrencyRateBean
DealBean	DeptBean
ItemBean	ItemSuppCountryBean
ItemSupplierBean	LocationBean
NonMerchCodeHeadBean	OrderBean
OrderLocationBean	PartnerBean
PeriodBean	ShipmentAPI
ShipmentBean	ShipSkuBean
StagePurgedShipmentsBean	StagePurgedShipskuBean
SupplierBean	SupTraitBean
SupTraitsMatrixBean	SystemOptionsBean
TermsBean	UpcBean
VendorBean	

RMS 10.1 referencing beans and APIs	
AddrBean	ClassBean
CodeDetailBean	CurrenciesBean
CurrenciesConversionAPI	CurrencyRateBean
DealBean	DeptBean
ItemBean	ItemSuppCountryBean
ItemSupplierBean	LocationBean
NonMerchCodeHeadBean	OrderBean
OrderLocationBean	PartnerBean
PeriodBean	ShipmentAPI
ShipmentBean	ShipSkuBean
StagePurgedShipmentsBean	StagePurgedShipskusBean
SupplierBean	SupTraitBean
SupTraitsMatrixBean	SystemOptionsBean
TermsBean	UpcBean
VendorBean	

Interface DAL porting example

As mentioned above, the DAL layer of ReIM is designed to support the replacement of the persistence mechanism that stores the actual data that is used to create the business objects used in the upper layers of ReIM.

As an example of this replacement process, this section illustrates the replacement of the AddrBean within ReIM. For the client, this example serves as a starting guide for the way to replace any general interface DAL object implementation within ReIM.

The AddrBean provides the data for the business object that represents a particular supplier's invoice address.

In general, the following steps provide the easiest method of modifying the ReIM interface DAL implementations:

- 1 Copy the interface DAL to your own java package.
- 2 Locate the SQL statements within the bean code and modify them to suit your own merchandising system.
- 3 Compile the new classes and place the class files into your http server environment.
- 4 Modify the beanDriver property in the com.retek.reim.reim.properties file to point to the new implementation.

ReIM interface DAL components

Properties file

The properties file `com.retek.reim.reim.properties` contains one property that is of importance when porting the ReIM interface DAL. The ‘beanDriver’ is the path to the applicable interface DAL beans. For example, if the client is using the standard RMS 9 interface beans, this property would be:

```
com.retek.reim.foundation.rms9
```

If your implementation requires either a DAL to another merchandising system or to another version of RMS, you should add a folder to the package structure (for example, `com.retek.reim.foundation.xyz8`) to hold the interface DAL beans. Reference this path as the beanDriver in your `com.retek.reim.reim.properties` file.

Interface DAL classes overview

The interface DAL is comprised of two types of classes: abstract beans and concrete implementation of the abstract beans.

The abstract beans provide the ‘signature’ of the interface DAL. That is, they provide the type information that is passed into a specific bean and the type of information that the service layer expects to be passed out of the bean. The abstract beans are referenced in the application code using the BeanFactory pattern. The BeanFactory checks to see which version of the concrete implementation should actually be called and makes the call.

The concrete implementations of abstract beans actually perform database operations. ReIM is shipped with concrete beans that provide interfaces to both base RMS 9 and base RMS 10.1. The properties file ‘beanDriver’ determines which set of queries is actually used.

If the client is porting the ReIM interface DAL to another version of RMS or to another merchandising system, the client need *not* change the presentation layer, service layer, or abstract bean code. Instead, the client must create concrete implementations of the abstract beans. These concrete beans must provide the correct information for the applicable merchandising system. Creating these concrete implementations is fairly simple if the client has a through knowledge of its schema, especially as it relates to the business functions of the merchandising system.

Abstract beans

The ReIM DAL abstract classes are located in the Java package `com.retek.reim.foundation`. These all follow the naming convention `ABusinessConceptBean.java` (meaning ‘A’bstact version of the Business Concept Bean). Each is a Java interface that declares the methods (and their parameters) that need to be provided by an implementation DAL class.

The following code represents a very simple example of the abstract bean, `AAddrBean.java`.

```
package com.retek.reim.foundation;

import com.retek.reim.merch.utils.ReIMException;

abstract public class AAddrBean
{
    abstract public long selectAddrKey(String vendor,
        String vendorType)
        throws ReIMException;
}
```

The `AAddrBean` provides the signature for any concrete implementations of the bean. The `AAddrBean` defines one method: `selectAddrKey`. The bean also defines the parameters of the method, returns a long and takes in a string that represents a vendor and a string that represents a vendor type. The `AAddrBean` does not have nor need any additional visibility as to how this information is actually retrieved.

This abstract `AAddrBean` is referenced in the application code using a `BeanFactory`. The `BeanFactory` package looks at the properties file and determines which concrete implementation of this bean should be used to actually retrieve data from the database.

Concrete implementations of abstract beans

Abstract beans do not actually access the database and retrieve data. Abstract beans provide the signature, and concrete beans actually do the work of interfacing with the database.

The RMS 9 standard implementation of the abstract `AAddrBean` (`com.retek.reim.foundation.AAddrBean`) is `AddrBean` (`com.retek.reim.foundation.rms9.AddrBean`). The concrete implementation of the bean is represented by the following:

```
package com.retek.reim.foundation.rms9;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

```

import java.sql.Statement;

import com.retek.reim.business.vendor.Vendor;
import com.retek.reim.foundation.AAddrBean;
import com.retek.reim.merch.utils.ReIMException;
import com.retek.reim.merch.utils.ReIMSeverity;
import
com.retek.reim.merch.utils.ReIMTransactionManager;

public class AddrBean extends AAddrBean
{
    public long selectAddrKey(String vendor, String
vendorType)
        throws ReIMException
    {
        long addrKey = -1L;
        Statement stmt = null;
        ResultSet rs = null;

        try
        {
            Connection conn =
ReIMTransactionManager.getConnection();

            stmt = conn.createStatement();

            String query =
                "SELECT ADDR_KEY FROM ADDR "
                + "WHERE (MODULE = 'SUPP' "
                + "AND KEY_VALUE_1 = '"
                + vendor
                + "') "
                + " OR (MODULE = 'PTNR' "
                + "AND KEY_VALUE_1 = '"
                + vendorType
                + "' AND KEY_VALUE_2 = '"
                + vendor
                + "') "

```



```

        + " AND ADDR_TYPE = '01' ";

        rs = stmt.executeQuery(query);

        while (rs.next())
        {
            addrKey = rs.getLong(1);
        }

        return addrKey;
    }
    catch (Exception e)
    {
        throw new ReIMException(
            "error.sql_error",
            ReIMSeverity.ERROR,
            e,
            this);
    }
    finally
    {
        try
        {
            if (rs != null)
            {
                rs.close();
            }
            if (stmt != null)
            {
                stmt.close();
            }
        }
        catch (Exception e)
        {
            throw new
com.retek.reim.merch.utils.ReIMException(
            "Error.sql_error",
            ReIMSeverity.ERROR,

```

```

        e,
        this);
    }
}

}

    public Long getDocumentAddressKey(String vendorType,
String vendorId) throws ReIMException
    {
        PreparedStatement stmt = null;
        ResultSet rs = null;

        String supplierQuery = "SELECT * FROM ADDR WHERE
MODULE = 'SUPP' AND KEY_VALUE_1 = ? AND ADDR_TYPE =
'05' ";

        String partnerQuery = "SELECT * FROM ADDR WHERE
MODULE = 'PTNR' AND KEY_VALUE_1 = ? AND KEY_VALUE_2 = ?
AND ADDR_TYPE = '05' ";

        try
        {
            Connection conn =
ReIMTransactionManager.getConnection();
            if(vendorType.equals(Vendor.SUPPLIER))
            {
                stmt = conn.prepareStatement(supplierQuery);
                stmt.setString(1,vendorId);
            }
            else
            {
                stmt = conn.prepareStatement(partnerQuery);
                stmt.setString(1,vendorType);
                stmt.setString(2,vendorId);
            }

            rs = stmt.executeQuery();
            if(rs.next())
            {
                return(new Long(rs.getLong("ADDR_KEY")));
            }
        }
    }
}

```

```

        }
        return null;
    }
    catch (Exception e)
    {
        throw new ReIMException(
            "Error.sql_error",
            ReIMSeverity.ERROR,
            e,
            this);
    }
    finally
    {
        try
        {
            if (rs != null)
            {
                rs.close();
            }
            if (stmt != null)
            {
                stmt.close();
            }
        }
        catch (Exception e)
        {
            throw new
com.retek.reim.merch.utils.ReIMException(
            "Error.sql_error",
            ReIMSeverity.ERROR,
            e,
            this);
        }
    }
}

    public boolean validAddressKey(String vendorType,
String vendorId) throws ReIMException

```

```

    {
        PreparedStatement stmt = null;
        ResultSet rs = null;

        String supplierQuery = "SELECT * FROM ADDR WHERE
MODULE = 'SUPP' AND KEY_VALUE_1 = ? ";
        String partnerQuery = "SELECT * FROM ADDR WHERE
MODULE = 'PTNR' AND KEY_VALUE_1 = ? AND KEY_VALUE_2 = ?
";

        try
        {
            Connection conn =
ReIMTransactionManager.getConnection();
            if (vendorType.equals (Vendor.SUPPLIER))
            {
                stmt = conn.prepareStatement(supplierQuery);
                stmt.setString(1,vendorId);
            }
            else
            {
                stmt = conn.prepareStatement(partnerQuery);
                stmt.setString(1,vendorType);
                stmt.setString(2,vendorId);
            }

            rs = stmt.executeQuery();
            if(rs.next())
            {
                return true;
            }
            return false;
        }
        catch (Exception e)
        {
            throw new ReIMException(
                "Error.sql_error",
                ReIMSeverity.ERROR,
                e,

```

```

        this);
    }
    finally
    {
        try
        {
            if (rs != null)
            {
                rs.close();
            }
            if (stmt != null)
            {
                stmt.close();
            }
        }
        catch (Exception e)
        {
            throw new
com.retek.reim.merch.utils.ReIMException(
            "Error.sql_error",
            ReIMSeverity.ERROR,
            e,
            this);
        }
    }
}
}

```

This concrete implementation of the bean actually accesses the database and gets information.

Because the main application code references the abstract bean, and the abstract bean (through the bean factory) references the concrete bean, only concrete beans need to be changed to interface ReIM with a pre-base 9 or a custom version of RMS or a totally different merchandising system. The distinct layering of the application architecture makes the changes needed to port to a different merchandising system very minimal.

Porting concrete beans

If the client's implementation requires either a DAL to another merchandising system or to another version of RMS, the client must create an applicable DAL.

For example:

RMS 8 holds supplier invoice addresses in the exact same manner as RMS 9. If the client wishes to implement ReIM against RMS 8, the client would set its `com.retek.reim.reim.properties beanDriver` to `com.retek.reim.foundation.rms8` and create the directory in the package structure. The client could then copy `com.retek.reim.foundation.rms9.AddrBean` to `com.retek.reim.foundation.rms8.AddrBean`. Because RMS 8 and 9 handle supplier invoice addresses identically, the client would then be finished porting the `AddrBean`. The client would also need to copy the RMS 9 versions of all other concrete interface beans into the RMS 8 bean driver directory. Some of these other beans (`ItemBean`, and so on) might have to be modified to make ReIM work with RMS 8 because of schema differences between versions of RMS.

The client would need to create a concrete bean for each of the abstract beans in the ReIM DAL. Depending upon the differences between either RMS 9 and RMS 10 and the system the client is implementing against, the client might be able to simply copy and paste all the beans into its new driver package. However, the client would most likely need to modify the SQL code (and not the Java) in some of these concrete beans to make them return the correct data from the merchandising system.

If the client defined its `com.retek.reim.reim.properties beanDriver` as `com.retek.reim.foundation.xyz8`, the client would need to create an `AddrBean` in the `com.retek.reim.foundation.xyz8` package structure. This `AddrBean` should do whatever is necessary to extract the invoice address for a supplier from whatever data structures hold this information. The `AddrBean` must take the vendor and vendor type as input parameters and return an address ID key as a long. The most likely changes to the `AddrBean` would occur in the SQL code.

As long as it extends the `AAddrBean` and therefore has the same signature, the `AddrBean` could perform any type of additional processing. The client must create concrete implementations for each abstract bean.

Summary of porting steps for custom merchandising systems

To implement ReIM with a previous version of RMS, customized version of RMS or a totally different merchandising system (in other words, a merchandising system other than RMS 9 or RMS 10.1), follow these high-level steps:

- 1 Determine the DAL data source.
- 2 Set the `com.retek.reim.reim.properties` beanDriver to the data source.
- 3 Create the directory specified as the beanDriver.
- 4 Copy either the RMS9 or RMS10 (depending on which is closer in schema to your merchandising system) beans into the beanDriver directory. Make sure that you have a concrete bean for each of the abstract beans.

Or

Copy some RMS9 concrete beans and some RMS10 concrete beans. Make sure that you have a concrete bean for each of the abstract beans.

- 5 Modify the SQL in the beanDriver concrete beans (as needed) to fetch the appropriate information from the data source.
- 6 Compile, test, and deploy.

Staging tables

The receiver cost adjustment and receiver unit adjustment interfaces to the merchandising system are handled via staging tables. These interfaces involve particularly complex business processes that are often modified by clients as part of RMS (or merchandising system-equivalent) implementations.

Because no standard path is satisfactory to all clients, ReIM does *not* attempt to execute all of the business processes in RMS (or its equivalent). Rather, ReIM writes pertinent information to staging tables, and the client is responsible for creating processes that use this information.

Receiver cost adjustment

ReIM writes a record containing the purchase order, item, location, old cost and corrected cost to a receiver cost adjustment staging table. In deciding upon how to create business processes that best fit its needs, the client has multiple choices.

One choice is to create a report that an online user can reference to create manual receiver cost adjustments in RMS (or its equivalent). This choice would allow for the execution of the custom logic that the client has built into the receiver cost adjustment dialog in RMS (or its equivalent).

A second choice is to automate the process by creating a program that reads from the staging table and performs all necessary receiver cost adjustment logic (including customizations) in RMS (or its equivalent).

Receiver unit adjustment

ReIM writes a record containing the shipment, purchase order, location, old quantity and corrected quantity to a receiver unit adjustment staging table. Again, in deciding upon how to create business processes that best fit its needs, the client has multiple choices.

One choice is to create a report that an online user can reference to create manual receiver unit adjustments in RMS (or its equivalent). This choice would allow for the execution of the custom logic that the client has built into the RMS (or its equivalent) receiver unit adjustment dialog.

A second choice is to automate the process by creating a program that reads from the staging table and performs all necessary receiver unit adjustment logic (including customizations) in RMS (or its equivalent).

Required merchandising system information

This section defines which RMS tables and columns are currently used by ReIM. It also shows which tables and columns are required for ReIM to function correctly.

If this client is using a merchandising system other than RMS (versions 9 or 10.1), equivalent tables and columns must be available for ReIM.

For a description of the RMS tables and columns, see the latest RMS data model.

DAL bean	RMS version	RMS table	Column	Selected in bean?	In bean where clause?	Notes
Addr	9 10.1	ADDR	Addr_key	X		
			Module		X	
			Key_value_1		X	
			Key_value_2		X	
			Addr_type		X	Specific where clause value: '02' (which stands for invoice address)
Class	9 10.1	CLASS	Dept	X	X	
			Class	X	X	
			Class_name	X	X	

DAL bean	RMS version	RMS table	Column	Selected in bean?	In bean where clause?	Notes
CodeDetail	9 10.1	CODE_DETAIL	Code_type	X	X	
			Code	X	X	
			Code_desc	X	X	
Currencies	9 10.1	CURRENCIES	Currency_code	X	X	
			Currency_desc	X		
			Currency_cost_fmt	X		
			Currency_retl_fmt	X		
			Currency_cost_dec	X		
			Currency_retl_dec	X		
CurrencyRates	9 10.1	CURRENCY_RATES	Effective_date	X	X	
			Exchange_rate	X		
			Currency_code		X	
			Exchange_type		X	Specific where clause value: 'C' (for the consolidated exchange rate)
Dept	9 10.1	DEPS	Dept	X	X	
			Dept_name	X	X	
			Buyer	X	X	
Item	9	DESC_LOOK	Sku	X		
			Desc_up	X		

DAL bean	RMS version	RMS table	Column	Selected in bean?	In bean where clause?	Notes
			System_ind		X	Specific where clause values: 'S', 'F', 'P' (meaning skus, fashion skus and pack items)
	10.1	ITEM_MASTER	Item	X		
			Item_desc	X		
			Tran_level		X	Specific where clause value: Tran_level = item_level
			Item_level		X	Specific where clause value: Tran_level = item_level
			Status		X	Specific where clause value: 'A' (meaning that the item is approved).
ItemSupplier	9	ITEM_SUPPLIER	Item	X		Includes a join to desc_look to also get the item desc
			Supplier	X		

DAL bean	RMS version	RMS table	Column	Selected in bean?	In bean where clause?	Notes
	10.1	ITEM_SUPPLIER	Item	X		
			Supplier	X		Includes a join to item_master to also get the item desc
Location	9	STORE	Store	X		
			Store_name	X		
		WH	Wh	X		
			Wh_name	X		
	10.1	STORE	Store	X		
			Store_name	X		
		WH	Wh	X		Specific where clause value: wh.wh = wh.physical_wh to ensure that only RMS10.1
			Wh_name	X		
NonMerch CodeHead	9 10.1	NON_MERCH_CODE_HEAD	Non_merch_code	X		
			Non_merch_code_head	X		
			Service_ind	X		
Ordloc	9 10.1	ORDLOC	Order_no	X	X	
			Location	X	X	
			Loc_type	X	X	

DAL bean	RMS version	RMS table	Column	Selected in bean?	In bean where clause?	Notes
Partner	9 10.1	PARTNER	Partner_id	X	X	
			Partner_type	X	X	
			Terms	X		
Period	9 10.1	PERIOD	Vdate	X		
Shipment	9 10.1	SHIPMENT	Shipment	X		
			Order_no	X		
			Asn	X		
			To_loc	X	X	
			Status		X	Specific where clause value: 'R' – received
			Inv_match_status		X	Specific where clause value: 'U', 'P' – unmatched and partially matched
Supplier	9 10.1	SUPS	Supplier	X	X	
			Sup_name	X		
			Terms	X		
			Currency_code	X		
SupTraits	9 10.1	SUP_TRAITS	Sup_trait	X		
			Description	X		

DAL bean	RMS version	RMS table	Column	Selected in bean?	In bean where clause?	Notes
SupTraitMatrix	9 10.1	SUP_TRAIT_MATRIX	Supplier	X		
			Sup_trait	X		
Terms	9	TERMS	Terms	X	X	
			Des	X		
			Duedays	X		
			Discdays	X		
			percent	X		
	10.1	TERMS	Terms	X	X	
			Terms_code	X		
			Terms_desc	X		
			Duedays	X		
			Discdays	X		
			Percent	X		
UPC	9	UPC_EAN	Upc	X		
			Upc_desc	X		
			Sku		X	
	10.1	ITEM_MASTER	Item	X		
			Item_desc	X		
			Item_parent		X	
			Item_level		X	
Vendor	9 10.1	SUPPLIER	Supplier	X	X	Supplier and partner tables are unioned together.
			Sup_name	X		

DAL bean	RMS version	RMS table	Column	Selected in bean?	In bean where clause?	Notes
		PARTNER	Partner_id	X	X	
			Partner_desc	X		
			Partner_type	X		

The financial system interface

ReIM has two types of financial interfaces: foundation financials data and transactional information. Both are described in this section.

Foundation financial data

The following types of financial information are imported in ReIM:

- Terms ranking data
- Variable department/class account segments
- Variable company/location account segments

Terms ranking information is used in the best terms calculation to choose the best term for each document. This best terms information is posted to the financial system.

Variable department/class and company/location segments are used to determine the account segments to which a document is posted.

ReIM provides an API for terms rankings, which are held in `TermsRanking.properties`.

The client is responsible for populating variable department/class and company/location segments. No API is provided.

Terms ranking

ReIM requires a file of terms rankings (held in `TermsRanking.properties`). The client is responsible for creating this file in the prescribed format. An ReIM process writes the terms ranking data to the ReIM tables. The terms ranking table (`IM_TERMS_RANKING`) stores the rankings.

A flat file is used to populate the table with the terms ID and the terms ranking. Terms ranking files should be uploaded to ReIM on a periodic basis.

This terms ranking interface is slightly more sophisticated than the other foundation financial data interfaces because terms are related to other parts of the system. Validation is performed to ensure that all terms IDs are valid and that there is a ranking for each term.

Location account segments

ReIM uses location account segments in general ledger (GL) account mappings. ReIM does *not* provide an interface for this information because it does not directly relate to other information in ReIM. ReIM expects the client to directly populate the ReIM location account segments table and keep it in sync with the financial application.

Department/class account segments

ReIM uses department account segments in GL account mappings. ReIM does not provide an interface for this information because it does not directly relate to other information in ReIM. ReIM expects the client to directly populate the ReIM department account segment table.

Financial transactions

To be independent of any single financial product, such as Oracle Financials Retek has created a generic interface. ReIM writes transactions that must be sent to financials to a financials staging table. That is, Retek writes records to a single generic table from which custom client code can read records and process data as necessary. The client is responsible creating a process that sends transactions to the financials system.

Resolution posting

To understand the process that posts data from ReIM to the financials staging table (IM_FINANCIAL_STAGE), see the section ‘Resolution posting action rollup’ in Chapter 7, “Batch processes”.

Major tables

- IM_TERMS_RANKING
- IM_DYNAMIC_SEGMENT_DEPT_CLASS
- IM_DYNAMIC_SEGMENT_LOC

LDAP and other user interfaces

There are two types of user authentication supported in ReIM: LDAP and database. A simple switch in the `reim.properties` file instructs the application as to which method to utilize. See Chapter 2, “Initial implementation and system parameters”.

LDAP

LDAP stands for Light Directory Access Protocol. The LDAP standard defines a network protocol for accessing information in a directory.

LDAP is one of the means of user authentication that ReIM supports. If it is used, LDAP is only used within ReIM for user authentication. Because ReIM has specific requirements for ReIM user roles and permissions that are easily client configurable, these are defined in the application itself. ReIM reads standard user information from an LDAP server.

If the client already stores user information using LDAP, the only interfacing configuration that needs to be done is in an LDAP-specific properties file. The entries in this file point ReIM to the appropriate machine, port, and so on to find the LDAP server. Other properties may need to be modified to reflect the names of attributes that the client uses in its LDAP schema.

Additional LDAP resources

- <http://www.openldap.org/>
This site contains the OpenLDAP main page. This site contains introduction, downloads, and documentation.
- <http://developer.novell.com/ndk/doc/jldap/>
This Novell site includes some additional Java classes. ReIM does not utilize these packaged classes.
- <http://www.iit.edu/~gawojar/ldap/>
This site is the LDAP browser site.
- <http://ldap.akbkhome.com/>
This site contains an LDAP schema view with some definitions of the standard LDAP object classes and attributes.

ReIM user table

A client that does not use LDAP has the option of entering valid users into the ReIM user table. Note, however, that ReIM does *not* provide any method for inserting user information into the ReIM user table. The client is responsible for this interface associated with user information.

Chapter 6 – Technical design

This chapter contains information related to the technical design of ReIM.

Services

Administration and configuration-related services

PreferenceService

- Returns either a named view of user preferences for a given user for a specific ReIM screen page or the default view for that screen page.
- Registers the active set of user preferences as a named view for a given user for a specific ReIM screen page.
- Returns all named views for a given user.

ReIMSystemOptionsService

Supports the configuration of and access to the following ReIM system-wide options:

- DocumentHistoryDays
- DebitMemoSendDays
- CloseOpenReceiptDays
- CostResolutionDueDays
- QtyResolutionDueDays
- PostDatedDocDays
- DebitMemoPrefixCost
- DebitMemoPrefixQty
- CreditMemoPrefixCost
- CreditMemoPrefixQty
- CreditNotePrefixCost
- CreditNotePrefixQty
- MaxTolerancePct
- DaysBeforeDueDate
- DaysBeforeDueDate
- DefaultPayNowTerms
- DefaultPayNowTermsDesc

Supports the configuration and access to the following auto-match options:

- CostResolutionDueDays
- QtyResolutionDueDays
- DaysBeforeDueDate

SavedViewService

- Creates, returns, or deletes named views of user preferences for a particular ReIM screen page.

SystemOptionsService

Supports the access to the following parameters:

- All system-wide options (see the ReIMSystemOptionsService above)
- Primary currency
- Primary language

Batch entry-related services

BatchEntryListService

- Returns a list of documents based on batch entry search criteria.
- Returns batch entries for a given batch group.
- Deletes existing batches (and associated invoices) for given batch group identifiers.
- Returns batch currencies.

BatchEntryNewService

- Validates documents entered in a given batch.
- Creates batch entry documents.
- Creates or updates document group list entries.
- Returns terms identifier from a given purchase order.
- Returns 'from terms' for a given terms identifier.
- Returns default currencies or default currency for a given currency code.
- Validates the existence of a document group list identified by a given document group identifier.
- Returns location type for a given location identifier.
- Calculates terms due date.

BatchEntryViewService

- Returns list of batch entry documents.
- Returns a set of batch entry documents for a given batch group identifier.
- Returns invoice type for a given document type.
- Deletes documents identified by the given document identifiers.
- Updates total cost and document member count control values for a given document group list identifier.
- Updates the status of a given document group list.
- Validates entered batch entry control totals and determines variance with calculated totals.
- Manages locking of batch entry information in the database.

CommentService

- Creates or returns comments for a given document or document line.
- Returns document line comments relating to a specific discrepancy (reason code).
- Returns document line comments relating to a specific debit reason code.

DataTranslationService

- Retrieves the ISO language code from the designated merchandising system.
- Translates data to the languages represented by the values of the following parameters:
 - UserLanguage
 - PrimaryLanguage
- Sets these ReIM language options:
 - ItemDescriptionLangOption
 - LocationNameLangOption
 - SupplierNameLangOption

Where UserLanguage is not equal to PrimaryLanguage individual language settings for ItemDescription, LocationName and SupplierName (multiLangOption)

Where UserLanguage equals PrimaryLanguage, a single language setting for ItemDescription, LocationName and SupplierName (singleLangOption)

DealService

- Returns the graduated purchase order cost for a given item on a given purchase order for a given location.
- Returns an item from a given purchase order for a given location (specified in the designated merchandising system).
- Returns the total discount for the given deals.
- Returns the projected purchase order cost for the given deals.
- Verifies purchase order deal existence for an item on a given purchase order for a given location from the designated merchandising system.
- Verifies the invoice deal's existence for an item on a given purchase order for a given location from the designated merchandising system.
- Returns supplier for a given purchase order.

Discrepancy-related services

DiscrepancyService

- Returns discrepancy 'resolve by' date.
- Archives resolved discrepancies to historical data storage.
- Archives cost and quantity discrepancies to historical data storage.
- Returns cost discrepancies from historical data storage.
- Returns cost and quantity discrepancies associated with a given document line item.
- Associates the business role of the user who created a given discrepancy with the discrepancy.
- Creates and deletes cost and quantity discrepancies.

VarianceResolutionService

- Creates a variance resolution and associates the resolution with a given document line item.
- Creates comments pertaining to a given resolution.
- Supports the reassignment of review responsibility for a given type of discrepancy to a given review group.
- Determines whether a given cost or quantity discrepancy variance is within allowed tolerances for that type of discrepancy.
- Creates adjustments to offset an identified variance.

Disputed credit memo-related services

DisputedCreditMemoResolutionRollupService

- Rolls up all the actions taken to resolve a disputed credit memo cost or credit memo quantity that was created as a reversal to a debit memo. The rollup occurs only if all lines on the disputed credit memo have been resolved (that is, either denied or approved). When the disputed credit memo was first created as a reversal to a debit memo, the original debit memo reason codes were associated with the new credit memo in disputed status. After the rollup, a new set of detail records is written for the credit memo. New reason codes for resolving the disputed lines will replace the old ones associated with the original debit memo. The credit memo is set to 'approved' status. The lines that are approved are rolled up to calculate the header level total cost and total quantity. Non-merchandise costs can be associated with a credit memo that is created as a debit memo reversal, but no resolution actions can be taken on non-merchandise costs. Non-merchandise costs should be included in the credit memo's total cost.

DisputedCreditMemoResolutionService

- Persists all unit cost and quantity resolutions to the database.

Document-related services

DocumentDetailService

- Creates new document detail reason code rows for the given set of items.
- Deletes the last document detail reason code row for the given set of items.
- Calculates total quantity for the given set of document items.
- Calculates total cost for the given set of document items.
- Creates a full document for the given document maintenance object.
- Creates a full document for the given memo data object.
- Creates a new document detail item and updates document header control values.
- Returns document item reason codes for a given document identifier.
- Deletes document detail reason code rows meeting the given criteria.
- Returns UPC and UPC Supplement for a given purchase order/item pair (supports the EDI invoice download process).

DocumentService

- Updates the status of the given set of documents.
- Returns document header information for a given document identifier.
- Returns batch vendors for a given vendor type.
- Returns document type description for a given document type identifier.
- Returns the document identifier prefix for the given document type.
- Returns the document status description for the given document status.
- Returns the document status for the given document status description.
- Returns the freight type description for a given freight type.
- Creates a document of the given document type.
- Returns the best term source for a given document source.
- Returns the best term date source for a given document date source.
- Returns document details for a given document identifier.
- Returns external document identifier for a given document identifier.
- Returns document identifier for a given external document identifier.
- Validates that external document identifiers are unique for a given vendor/document type.
- Manages the locking of document information in the database.
- Returns total allowances for a given document identifier.
- Returns document match dates for a given document identifier.
- Returns approval date for a given document identifier.
- Returns the set of matched invoices ready for posting.
- Returns approved credit notes, credit memos, debit memos and credit note requests.
- Returns late credit note requests.
- Returns all invoices for the given list of purchase order locations.
- Returns order information for a given document identifier.
- Updates the given set of invoices to 'Pre-Paid'.
- Returns the allowance code amount for a given document/item pair.
- Returns the reference document for a given document identifier.
- Deletes the document identified by the given document identifier.
- Returns the exchange rate for the given purchase order/vendor/vendor type/document date.
- Returns approved non-merchandise invoices.

EDI-related services

EdiDocumentService

- Creates an EDI document given an EDI transaction header.

EdiRejectService

- Returns EDI reject document details.
- Returns EDI reject documents.
- Deletes EDI reject documents with the given document identifiers.
- Returns suppliers.
- Returns items from a given supplier.
- Returns items for the given supplier/item pair.
- Returns purchase orders for the given supplier identifier..
- Manages locking for EDI rejects in the database.
- Allows mass deletion of EDI rejects for a given purchase order.
- Validates EDI document values.
- Validates that the invoice date is within range.
- Creates new and updates rejected EDI documents.

EdiTranslationHeaderValidationService

- Validates post-dated document days.
- Validates the given term identifier.
- Validates the vendor document date.
- Creates EDI reject documents.

Financial-related services

DebitMemoReversalService

- Creates, returns, updates or deletes debit memos.
- Returns reversed debit memo header.
- Returns reversed debit memo details.
- Returns reversed non-merchandise costs.
- Routes cost or quantity discrepancies.
- Creates cost or quantity discrepancy routings.
- Returns the document total merchandise cost.
- Reverses memo cost or memo quantity.
- Returns reversed cost.

GLAccountService

- Creates, returns, updates, or deletes GL options.
- Creates, updates or deletes GL cross references.
- Returns GL cross references for the given account type/account code pair.
- Returns GL account segments.
- Returns distinct account codes for the given account type.
- Defines GL account segments for location and company (based on receipt location).
- Defines GL account segments for class/department based on a sample item from the purchase order.

ResolutionPostingService

- Posts approved credit notes.
- Posts matched invoices.
- Posts approved non-merchandise invoices.
- Posts approved debit memos.
- Posts approved credit memos.
- Posts matched invoices for the given purchase order locations.
- Posts matched invoices from the given set of invoices.
- Posts approved non-merchandise invoices from the given set of invoices.
- Posts approved credit notes from the given set of credit notes.
- Posts approved debit memos from the given set of debit memos.
- Posts approved credit memos from the given set of credit memos.
- Posts receipt write-offs from the given set of receipt write-offs.
- Defines resolution reason code records.
- Converts late credit note requests.
- Posts pre-paid invoices for the given set of invoices.

Foundation-related services

UIFieldValidationService

- Identifies numbers as decimal or whole numbers.
- Compares two decimal numbers and indicates the relative size of each number.

Internationalization-related services

I18NService

- Returns currency locale.
- Returns primary currency code.

Invoice-related services

InvoiceAdvSearchService

- Populates a document list with results from a query based on the given search criteria.

InvoiceDetailService

- Populates invoice items for the given documents.
- Creates, updates or deletes invoice items for the given document/item pairs.
- Returns invoice items for the given invoice identifier.
- Returns invoice items from receipts with the given receipt identifiers.
- Returns invoice items for the given purchase order/item pairs.
- Verifies that invoice detail data exists for a given document identifier.

InvoiceMaintenanceService

- Validates total merchandise cost for merchandise invoices.
- Validates cost range.
- Validates total non-merchandise cost for non-merchandise invoices.
- Validates total merchandise cost for non-merchandise invoices.
- Validates total invoice quantity for merchandise invoices.
- Validates total merchandise cost and the sum of extended item cost for the given items.
- Validates total invoice quantity and the sum of item quantities for the given items.
- Validates that the invoice date is not before the Vdate.
- Validates that the invoice date is before the post-dated document days value.

ParentInvoiceService

- Returns parent invoices.
- Returns parent total non-merchandise cost for the given parent invoice identifier.
- Creates a parent invoice header for an invoice with multiple locations.
- Returns the parent invoice with the given parent invoice identifier value.
- Reloads split (child) invoices.
- Creates and deletes child invoices.
- Returns child invoices from a given parent purchase order.

- Returns undistributed cost from the given child invoices.
- Returns undistributed quantity from the given child invoices.
- Updates total merchandise cost and total merchandise quantity for the given child invoices.
- Deletes child invoices.
- Manages locking on parent invoice data in the database.

Matching-related services

AutoMatchMetricsService

- Creates AutoMatch metrics.

AutoMatchService

- Matches invoices with receipts for the given purchase order/location pairs.
- Separates purchase order locations into purchase order locations that have receipts and purchase order locations that do not. Purchase order locations with receipts will be processed by the receipt-matching algorithm. Purchase order locations without receipts will be processed by the cost pre-matching process.
- Matches invoice line unit cost with the unit cost on the corresponding purchase order line. This matching is only performed for purchase order locations without receipts. Creates cost discrepancies for invoice lines that do not match within tolerance.
- Routes all unreceived invoices.
- Saves the results of all successful summary or detail-level matches.
- Performs a purchase order location-level summary match for a given purchase order location. All candidate invoices and receipts associated with the purchase order location are examined. A match is considered successful if the total extended costs (and optionally, total quantities) of the associated invoices and receipts match within the specified tolerances. If the match is successful, all invoices and receipts associated with the purchase order location are set to 'matched' status; otherwise, the associated invoices and receipts are set to 'unresolved match' status.
- Performs a 'single invoice-to-single receipt match' at the summary level. This match is attempted only if the purchase order location-level summary match is unsuccessful. If a one-to-one match within tolerances is found for total extended costs (and optionally, total quantities) for the invoice and the receipt, the invoice and the receipt status are set to 'matched'. If one invoice can be matched to more than one receipt, or if one receipt can be matched to more than one invoice, the invoice and receipt status will remain in 'unresolved match' status, and a later process will put them into 'multi-unresolved' status.
- Verifies whether a purchase order location is in 'multi-unresolved' status.

- Returns invoices and receipts that are ‘unresolved’ after summary matching for a line-level (detail) match for a given purchase order location.
- Performs a line-level (detail) match for a given purchase order location.

DetailMatchService

- Builds detail item views from the given summary group unmatched invoice map and unmatched receipt map.
- Submits the given invoice items and receipt items for matching.
- Performs matching on the given invoice item group list and receipt item group list.
- Persists ‘in-balance’ matches for the given detail-level match items.
- Calculates unit cost in order to perform cost resolution on the given invoice and receipt items.

ManualGroupService

- Returns next unassigned group identifier.
- Validates the existence of a manual group with the given manual group identifier.
- Returns a set of manual groups for the given invoice and receipt identifiers.
- Creates a manual group.
- Creates, updates or deletes an invoice group for a given manual group identifier.
- Creates, updates or deletes a receipt group for a given manual group identifier.

MatchHistoryService

- Matches invoice/receipt costs or quantities.
- Determines if a given match is an exact match.

MatchService

- Returns matched status for a given purchase order location.
- Returns matched status for a given invoice.
- Returns matched status for the given receipts.
- Updates matched status for given invoices.
- Updates matched invoice line status for given invoices.
- Updates matched invoice line status for items on given invoices.

MatchStatusService

- Returns matched status for a given purchase order location.
- Returns matched status for a given invoice.
- Returns matched status for a given receipt.
- Updates matched status for given purchase order locations.
- Updates matched status for given invoices.
- Updates matched invoice line status for given invoices.
- Updates matched invoice line status for items on given invoices.

PriceReviewService

- Returns the cost review list.
- Returns price review detail.
- Returns the invoice review status given a document/item pair.

QuantityReviewService

- Returns a quantity review list.
- Returns a set of quantity discrepancies.
- Returns quantity variance and quantity variance percent for the given total receipt quantity/quantity invoiced values.
- Returns quantity discrepancy receipts.
- Returns receipt quantities.
- Returns purchase order comments for a given purchase order.
- Validates a given quantity value.
- Returns the resolution quantity for the given quantity discrepancy identifier.

ReceiverAdjustmentService

- Verifies whether or not receiver adjustment changes were made in the designated merchandising system. Once the changes have been validated, the Adjustment Pending attribute is set to "N". For receiver cost adjustment (RCA), the cost adjustment change in the order is checked. For receiver cost adjustment supplier (RCAS), the cost adjustment change in the order is checked, and the cost adjustment change for the supplier is checked. For receiver unit adjustment (RUA), the quantity adjustment change in the receipt (shipment) is checked.

SummaryMatchService

- Returns a summary match list based on the given search criteria.
- Returns set of suppliers based on the given summary match search criteria.
- Returns summary match candidate invoice and receipt lists.
- Returns purchase order locations for the given invoice and receipt lists.
- Generates summary automatic matching group map from the associated purchase order locations.
- Generates summary manual matching group map from the associated manual match group information
- Generates and returns lists of unmatched invoices and receipts.
- Returns the supplier description for a given supplier identifier.
- Returns the total non-merchandise cost of a given invoice at a given purchase order location.
- Calculates grouped summary matched invoice and receipt cost and quantity totals.
- Generates, saves, updates and delete summary matched invoice and receipt manual groups.
- Performs summary automatic invoice and receipt matching.
- Combines summary match automatic groups.

TermsRankingService

- Returns the current terms rankings from the TermsRanking.properties file.

ToleranceMaintenanceService

- Returns the supplier currency for a given supplier identifier.
- Returns tolerance information given one of the following: a department identifier, a supplier identifier, a supplier trait identifier or a corporate-level identifier.
- Creates, modifies or deletes (and audits the deletion of) tolerance information at one of the following levels: department, supplier, supplier trait, corporate.
- Determines the authorization of a given user to maintain tolerance information. Authorizations may be granted at the following levels: department, supplier, supplier trait, corporate.
- Applies tolerances to the appropriate level in the product hierarchy.
- Returns the set of suppliers for a given supplier trait.
- Assigns a given supplier trait to a default set of suppliers.
- Returns any coverage gaps or overlaps for a given set of tolerances.

ToleranceService

- Populates AutoMatch tolerances.

VarianceService

- Calculates summary variance for matched documents and determines if the variance is within tolerance.
- Calculates detail variance for matched items and determines if the variance is within tolerance.
- Calculates cost and quantity variance.
- Calculates extended cost variance.

Merchandising system-managed data

ClassService

- Returns item class from the designated merchandising system.

CodeDetailService

- Returns code and code description from or validates code value against the codes maintained by the designated merchandising system.

CurrencyService

- Returns the consolidated exchange rate from the designated merchandising system.
- Validates the currency code value against the codes maintained by the designated merchandising system.
- Converts the currency code value to other currency codes maintained by the designated merchandising system.
- Returns the vendor currency code.
- Returns the purchase order currency code from the designated merchandising system.

DepartmentService

- Returns the department name for the given department identifier from the designated merchandising system.

ItemService

- Validates a UPC or UPC Supplement from the designated merchandising system.
- Returns item description for a given item identifier from the designated merchandising system.

ItemSupplierService

- Returns items for a given supplier identifier from the designated merchandising system.
- Returns invoice detail items for a given supplier identifier from the designated merchandising system.
- Returns invoice detail items for the given supplier identifier/partial item description pair from the designated merchandising system.
- Returns document detail items for a given supplier identifier from the designated merchandising system.
- Returns document detail items for the given supplier identifier/partial item description pair from the designated merchandising system.
- Validates an item for a given supplier from the designated merchandising system.

LocationService

- Validates store from the designated merchandising system.
- Validates warehouse from the designated merchandising system.
- Returns location description for the given location identifier from the designated merchandising system.

OrderService

- Returns purchase order terms for the given purchase order from the designated merchandising system.
- Validates the given purchase order using the designated merchandising system.
- Validates the linked suppliers for the given purchase order using the designated merchandising system.
- Returns purchase order defaults for the given purchase order from the designated merchandising system.
- Returns purchase order items for the given purchase order from the designated merchandising system.
- Returns purchase order status.

PartnerService

- Creates a partner of the specified partner type.
- Returns the partners for a given vendor from the designated merchandising system.

PeriodService

- Returns the VDate from the designated merchandising system.

TermsService

- Returns the terms information from the designated merchandising system for a given term identifier.
- Sets the best terms for given merchandise document(s) based on a given terms ranking.
- Sets the best terms for matched invoices at the given purchase order locations.
- Calculates the receipt of goods date for the given invoice(s) and receipt(s).
- Sets the best terms for the given detail-matched invoices(s).
- Returns the term description from the designated merchandising system for a given term identifier.

VendorService

- Returns the following from the designated merchandising system:
 - Vendor
 - Vendor currency
 - Document address identifier
 - Partner description
 - Supplier description
 - Payment terms identifier for the given vendor identifier
- Initializes a new vendor.

Non-merchandise document-related services

NonMerchandiseDocumentService

- Returns total non-merchandise cost for a given document identifier.
- Creates or deletes a non-merchandise document given a document identifier.

NonMerchEntryService

- Creates or returns non-merchandise document(s) for a given document identifier.
- Returns non-merchandise codes.
- Returns a non-merchandise code description for a given document identifier.
- Returns non-merchandise invoice identifiers.

NonMerchService

- Validates a given non-merchandise code.
- Returns non-merchandise document header for the given non-merchandise code from the designated merchandising system.

Reason codes-related services

ReasonCodeActionRollupService

- Rolls up credit memos and debit memos created from discrepancy resolutions.

ReasonCodesService

- For a given identifier, validates the existence of a business role with that identifier value.
- Returns a set of reason codes.
- Returns reason codes for actions that require GL account distribution.
- Returns a set of reason codes for a given action code.
- Returns a set of reason codes for a given business role/action code.
- Returns a reason code description for a given reason code.
- Returns a reason code hint comment for a given reason code.
- Returns all reason codes.
- Manages locking for reason code information in the database.
- Posts reason codes.
- Returns reason code actions for a given reason code.

Receipt-related services

ReceiptService

- Returns receipts for a given invoice/purchase order pair.
- Validates receipt existence for invoices meeting the criteria in the given where clause.
- Returns receipt matching candidate documents for the given receipt identifier.
- Validates that receipt matching candidate documents are ready for matching.
- Calculates available to match quantity for a given receipt item.
- Returns item quantity received for a given receipt/item pair.
- Validates whether a given receipt is unmatched.

ReceiptWriteOffService

- Returns receipt write-offs for purged or closed receipts.

Supplier-related services

SupplierOptionsService

- Returns system options from the designated merchandising system.
- Returns primary currency from the designated merchandising system.
- Returns primary language from the designated merchandising system.

SupplierService

- Returns the terms from the designated merchandising system for a given supplier identifier.
- Returns the linked suppliers from the designated merchandising system for a given supplier identifier.
- Returns the supplier description from the designated merchandising system for a given supplier identifier.
- Creates and deletes linked suppliers.

User roles-related services

UserRoleMaintenanceService

- Creates business roles. Returns, updates and deletes business roles for a given role identifier.
- Validates business roles existence for a given business role name.
- Creates or returns sets of user role members for a given business role identifier.
- Manages locking of user role information in the database.
- Returns a set of all users or user information for a given user identifier.
- Returns business role information for a given user identifier.
- Creates or returns department/class information for a given business role.
- Creates business role location information or returns business role locations for a given business role identifier.
- Creates business role reason code information or returns business role reason codes for a given business role identifier.
- Returns locations from the designed merchandising system.

UserRoleService

- Returns user role for a given item identifier.
- Returns user role identifier for a given location.

Locking design summary

ReIM locking is accomplished using database tables that hold record level locks. The locking of tables is performed for several reasons, including the following:

- ReIM does not necessarily maintain a single connection throughout an entire screen/process. That is, the system opens a connection, fetches information, and then closes the connection. At a later moment in time, the system opens another connection to save changes and close the connection.
- ReIM cannot maintain locks in some kind of Java session structure because the system may be involved with more than one Java virtual machine (JVM).

Locking and tables

Base tables that contain information to be locked (for example, IM_SUPPLIER_OPTIONS) have a corresponding ..._LOCK table (for example, IM_SUPPLIER_OPTIONS_LOCK). The ..._LOCK table contains the same columns as the primary key of the base table.

When the system creates a lock, it writes the primary key values for the base table records to be locked to the appropriate ..._LOCK table. For example, if data in the IM_SUPPLIER_OPTIONS table is to be locked for supplier '12345', a record is written to the IM_SUPPLIER_OPTIONS_LOCK table for supplier with the primary key value '12345'.

When records in a base header table are locked, all detail records related to each locked header record are implicitly locked. Detail records are not explicitly locked because:

- ReIM functionality must 'go through' the header information to access detail information. In other words, the entry point to detail records is generally through the header.
- On screens and within backend processes that include header information, some kind of summary of the details also exists.

The following two examples represent this type of header detail locking:

Example 1

If user A is looking at the header, and user B changes the details, user A does not have visibility to the changes and might perform an invalid action. Invoices are stored on IM_DOC_HEAD, and the non-merchandise costs on invoices are stored on IM_DOC_NON_MERCH. On the invoice header screen, user A can see a sum of all of the non-merch costs for invoice 99999. If user B could somehow at the same time add new non-merchandise costs for invoice 99999, the information that user A sees as the summary of non-merchandise costs would be invalid.

Example 2

If auto-match has selected all documents 'ready for match' and is processing and then additional data is entered for a document, the details with which the auto-match is working would no longer be valid.

Locking management

- When a user that has an active lock exits a screen (that is, the user presses OK or Cancel buttons on the screen), data changes are committed (if necessary) and then any locks on data displayed on that screen are removed. If any expired locks on the screen data exist, they are also released upon screen exit.
- When a user tries to commit information to the database, the locking service checks to ensure that the user has valid locks on any changed data being committed (for example, locks could have timed out as noted below). If the user does *not* have valid locks, the user receives a message noting that the user's changes *cannot* be saved. In this case, the user must exit the screen, enter the screen again, and re-enter the data changes that could not be committed due to invalid/expired locks.
- In situations where accidental system exits occur (for example, the server shuts down unexpectedly from power loss), locks are *not* released immediately. After the system is restored from outage, the user will log into the system and access the main menu. At that point, any existing data locks are removed. Because this data is no longer locked, any user with adequate security permissions can acquire new locks on this data.
- The lock timeout interval is defined in the `reim.properties` file. See Chapter 2, "Initial implementation and system parameters".
- When locks are written to the `..._LOCK` table, they include an 'end time' value. When checking to see if a row of data is locked, the system inspects the related lock row 'end time' value. If the commit time is before the end time on the `..._LOCK` table record, the base table data changes may be committed. If the commit time is equal to or exceeds the end time, the data lock will be treated as 'expired' and the data changes will not be committed.
- If a user needs immediate access to already locked data and cannot wait for data locks to expire or be released by the user holding the locks, a database administrator can manually delete existing lock records from the appropriate `..._LOCK` table to release the locks. However, this does not guarantee that the user that needs immediate access will be the next user to acquire locks on the just-released data. The manual release of locks should be a rare event due to the other lock release methods in the system.

Currency design summary

ReIM has been designed to handle a multiple number of currencies. This section addresses the system's assumptions, conversion process, and validations that are related to this capability.

Merchandising system (such as RMS) and ReIM assumptions

- RMS defines one currency as the primary currency of the system (held on the RMS SYSTEM_OPTIONS table in the CURRENCY_CODE field).
- RMS specifies that each purchase order can have one currency. This purchase order currency does *not* have to be the same as the RMS primary system currency or the RMS supplier currency.
- ReIM requires that each document have its currency stated (im_doc_head.currency_code). This invoice currency does *not* have to be the same as the system primary currency.
- ReIM assumes that a purchase order and any invoices associated with that purchase order are in the same currency. This assumption is based on the business reality that these currencies are almost always the same and on the development consideration that currency conversion processes have an adverse impact on system performance.

Currency conversion process for amount tolerances

- Amount tolerances are established in the primary currency of the system. However, because the invoices and POs to be matched could reflect a different currency, amount tolerances must be converted before they can be applied. In other words, the currency established for amount tolerances is converted when the invoice/PO combination is not in the primary currency of the system. For example, a tolerance defined as 10 US dollars (USD) has a much different meaning than a purchase order/invoice defined in Thai Bhat (10 Thai Bhat is about 0.23 USD). If the system merely utilized the number 10 and failed to perform a currency conversion, the amount tolerances would not apply correctly.
- Currency conversion rates are stored on the RMS CURRENCY_RATES table. The conversion factors on this table are in terms of the primary currency of the system.

For example, let's say a client wishes to convert from Thai Bhat to Uruguayan Pesos and that the system's primary currency is USD. First, the system performs a conversion from Thai Bhat to USD. Secondly, the system converts the USD value to Uruguayan Pesos. In other words, to perform its conversions, the system always must 'go through' the primary currency of the system.

Currency-related system validations

- One of the validations performed by the EDI upload process is that it determines whether the currency on the invoice is the same as the currency on the purchase order. If the invoice currency is *not* the same as the purchase order currency, the invoice is rejected.
- The graphical user interface (GUI) invoice entry (both single invoice entry and batch invoice entry) process also validates that the currency on the invoice is the same as the currency on the PO associated with the invoice. If the currencies are not the same, the user receives a warning message.

Java currency formatting

Localization, also known as L10N, is the process of adapting software that has been internationalized so that it can be released into a local market with its own language. Currency must be properly formatted according to its applicable locale. For example, US currency uses a comma as a thousands separator whereas other currencies do not use a comma as a thousands separator. Java has built-in libraries for currency formatting that are based on locales.

ReIM uses built-in Java localization functionality mapped through the table `IM_CURRENCY_LOCALE` to RMS's existing currency structure. ReIM provides an installation script that populates this table. The script creates records for every currency that RMS supports. Note that ReIM cannot guarantee the accuracy of RMS's language data.

Chapter 7 – Batch processes

This chapter provides the following:

- An overview of the batch architecture
- A functional summary of each batch processes, along with its dependencies
- A description of some of the features of the batch processes (batch return values, batch threading, and so on)
- Development designs for each batch process

Batch architectural overview

ReIM's batch processes are run in Java. Batch processes engage in their own primary processing. However, they utilize services when they must engage in actions outside their primary processing (for example, when they utilize a helper method, touch the database, and so on).

Services retrieve the data on which the batch processes 'work' to complete their tasks. As noted in Chapter 3, "Technical architecture", the service layer consists of a collection of Java classes that implements business logic (data retrieval, updates, deletions, and so on) via one or more high-level methods.

The business logic occurs within the service code, while the technical processing occurs within the batch code.

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

- They are not accessible through a graphical user interface (GUI).
- They are scheduled by the client.
- They are designed to process large volumes of data.
- They should only be executed during 'off-hours' (that is, during a time when users are not in the system such as nights).

EDI-related file-based batch processes

ReIM's EDI-related batch processes are file based. For example, they either input a flat file into the system (EDI invoice upload) from outside the system, or they output a flat file from the system (EDI invoice download) to be sent to another system (that of a vendor). Both the EDI invoice upload and the EDI invoice download batch processes are described later in this chapter.

Internal batch processes

Other batch processes within ReIM do not input or output files. Rather, the goal of these batch processes is to take a snapshot of potentially large amounts of data from the key tables within the database, transform that data through processing, and then return it.

These batch processes are located within the code ‘close’ to where they functionally fit. Thus, for example, the batch process called Auto-match (AutoMatchService) resides within the package: `com.retek.reim.services.matching` because its processing is the invoice matching functionality.

Internal batch processes that are described later in this chapter include:

- Auto-match
- Batch purge
- Disputed credit memo action rollup
- Reason code action rollup

Internal batch processes that write to staging tables

The third type of batch process within ReIM takes a snapshot of potentially large amounts of data from the key tables within the database, transforms that data through processing, and then writes that data to staging tables.

This communication process has been designed with the assumption that, during production, ReIM will reside within the same database as the merchandising system. Presumably, during implementation, the client will develop an optimum way to move the applicable data from the staging table(s) to the appropriate location for that data.

The internal batch processes that write to staging tables are described later in this chapter. They include the following:

- Resolution posting
- Receiver adjustment

Batch names and Java packages

The following table describes ReIM’s batch processes and their associated Java packages. The table’s order reflects the dependencies that exist among the ReIM batch processes but does not include any dependencies that exist between ReIM and the merchandising system it interacts with.

Batch name	Batch process	Package
Batch purge	BatchPurge	<code>com.retek.reim.purge</code>
EDI invoice upload	Ediupinv	<code>com.retek.reim.batch</code>
Receiver adjustment	ReceiverAdjustmentService	<code>com.retek.reim.services</code>
Auto-match	AutoMatchService	<code>com.retek.reim.services.matching</code>
Reason code action rollup	ReasonCodeActionRollupService	<code>com.retek.reim.services</code>
Disputed credit memo action rollup	DisputedCreditMemoResolutionRollupService	<code>com.retek.reim.services</code>

Batch name	Batch process	Package
Resolution posting	ResolutionPostingService	com.retek.reim.services
EDI invoice download	EdiDownload	com.retek.reim.batch

Functional descriptions and dependencies

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

Batch processes	Details	Batch dependencies
Batch purge (BatchPurge)	The batch purging process (BatchPurge.java) deletes data from database tables while maintaining database integrity. This process deletes records from the ReIM application that meet certain business criteria (for example, records that are marked for deletion by the application user, records that linger in the system beyond certain number of days, and so on).	
EDI invoice upload (ediupinv)	This batch process uploads merchandise and non-merchandise invoices and credit notes from the EDI into the invoice-matching tables.	

Batch processes	Details	Batch dependencies
Receiver adjustment (ReceiverAdjustmentService)	The process compares the unit cost and/or quantity received for the item on the shipment with the expected unit cost and/or quantity on the IM_RECEIVER_COST_ADJUST and/or IM_RECEIVER_UNIT_ADJUST tables. If a match exists, the receiver cost and/or unit adjustment has occurred in RMS (or the equivalent merchandising system). As a result, the process sets the 'pending adjustment' flag on IM_INVOICE_DETAIL table to false for the invoice line. The reason code actions are only rolled up for an invoice if no invoice lines on the invoice have any pending adjustments.	The receiver adjustment needs to be run after a client-written check for Receiver Cost Adjustments (RCAs) and and Receiver Unit Adjustments (RUAs).
Auto-match (AutoMatchService)	Auto-match is a system batch process that attempts to match invoices to receipts without manual intervention. Invoices that are in ready for match, unresolved, or multi-unresolved status are retrieved from the database to be run through the auto-match algorithm. The processing consists of two levels – summary and detail.	EDI upload (Invoice Matching) Receipt upload (Merchandising system, such as RMS)

Batch processes	Details	Batch dependencies
Reason code action rollup (ReasonCodeActionRollupService)	This batch process sweeps the action staging table and creates debit and credit memos as needed. Only a single debit or credit memo is created per invoice, with line details from all related actions. This process deletes these records when completed; they are deleted after posting. Note that a separate, client-created batch process sweeps the receiver adjustment table. The action staging table is used during posting to post the reason code actions to the financial staging table.	Receiver adjustment must occur prior to this batch process.
Disputed credit memo action rollup (DisputedCreditMemoResolutionRollupService)	The disputed credit memo action rollup process checks the records on the IM_REVERSAL_RESOLUTION_ACTION table and rolls up the credit memo detail lines by document/item/reason code. The rollup occurs only if all lines on a disputed credit memo have been completely resolved (that is, no cost or quantity discrepancy records remain for the credit memo). After the rollup, a new set of detail lines associated with the resolution reason codes replace the original set of detail lines associated with the debit reason codes on the IM_DOC_DETAIL_REASON_CODES table.	The disputed credit memo action rollup must occur before resolution posting and after receiver adjustment.

Batch processes	Details	Batch dependencies
Resolution posting (ResolutionPostingService)	<p>A recurring resolution posting process retrieves all matched invoices and approved documents.</p> <p>For each invoice, the batch process engages in the following high-level steps:</p> <ol style="list-style-type: none"> 1. Performs any resolution actions (for example, instigates the creation of payment documents). 2. Calls the posting process to write applicable financial accounting transactions to the financials staging table, IM_FINANCIALS_STAGE. 	
EDI invoice download (EdiDownload)	<p>The EdiDownload module creates a flat file to match the EDI invoice download file format. The module retrieves all header, detail and non-merchandise information and formats the data as needed.</p> <p>In other words, the EDI invoice download process retrieves debit memos, credit note requests, and credit memos in 'approved' status from the resolution posting process and creates a flat file. The client converts the flat file into an EDI format by the client and sends it via the EDI invoice download transaction set.</p>	Auto-match must run prior to the EDI invoice download.

Features of the batch processes

Scheduler and the command line

If the client uses a scheduler, batch process arguments are placed into the scheduler.

If the client does *not* use a scheduler, batch process parameters must be passed in at the Unix command line.

The Java batch processes are to be called via the shell scripts ‘reimautomatch’, ‘reimediinvdownload’, ‘reimediinvupload’, ‘reimposting’, ‘reimpurge’, and ‘reimrollup’. Each of these scripts interacts with the ‘generic’ shell script. These scripts take any and all arguments that their corresponding batch process would take when executing.

Batch return values

The following guidelines describe the batch process return values that ReIM’s batch processes utilize:

- SUCCESS = 0
- FAILED_INIT = 1
- FAILED_PROCESS = 2
- FAILED_WRAPUP = 3
- SUCCESS_WITH_REJECTS_TO_DB = 4
- SUCCESS_WITH_REJECTS_TO_FILE = 5
- SUCCESS_WITH_REJECTS_TO_DB_AND_FILE = 6
- UNKNOWN = -1

Batch log and error file paths

Log file locations are determined by the client via the reim.properties file. If an error occurs that causes a batch process to suddenly come to a complete halt, the system writes to the error file. See Chapter 2, “Initial implementation and system parameters”.

A note about multi threading

The following two batch processes shown below have multi-threading capabilities. The settings related to the multi-threading options for each batch process are established in the reim.properties file. See Chapter 2, “Initial implementation and system parameters”.

EDI invoice upload (ediupinv)

This process is threaded by each transaction in the file (THEAD record to TTAIL record). Each thread handles transaction validation and insertion into the database (as valid or rejected) or facilitates the writing to a reject file.

Auto-match (AutoMatchService)

Auto-match can either be run as a single thread or it can be threaded by the location hierarchy.

A note about restart and recovery

Most ReIM batch processes do *not* utilize any type of restart and recovery procedures. Rather, if a restart is required, the process can simply be restarted, and it will start where it left off. This solution is true for all batch processes other than the EDI invoice upload and the EDI invoice download (their restart and recovery methods are described in their designs below).

Batch purge

Overview

The batch purging process (BatchPurge.java) deletes data from database tables while maintaining database integrity. This process deletes records from the ReIM application that meet certain business criteria (for example, records that are marked for deletion by the application user, records that linger in the system beyond certain number of days, and so on). The BatchPurge process does not generate any cascade relationships and/or SQL queries on the fly. The main features of the process are illustrated below:

Usage

The following command runs the BatchPurge job:

```
BatchPurge userid/password PURGE [ALL|<table name>]
[NOCOMMIT|COMMIT]
```

Where the first argument is a combination of user id and password. The second argument is the word PURGE. The third argument is either ALL or a single table name. Table name can be any one of the following:

- 1 IM_DOC_GROUP_LIST
- 2 IM_DOC_HEAD
- 3 IM_PARENT_INVOICE
- 4 IM_REASON_CODES
- 5 IM_PARTIALLY_MATCHED_RECEIPTS
- 6 IM_TOLERANCE_DEPT_AUDIT
- 7 IM_TOLERANCE_SUPP_AUDIT
- 8 IM_TOLERANCE_SUTRT_AUDIT
- 9 IM_TOLERANCE_SYS_AUDIT

ALL deletes data from all of the above tables. Finally, the fourth argument can be either NOCOMMIT or COMMIT. If there is no fourth argument, the default is NOCOMMIT.

SQL queries

Delete statements have been optimized by minimizing the usage of nested SELECT statements and by maximizing the ‘table joins’ in the WHERE clause. Any additions and/or modifications to the database require manual additions and/or modifications, respectively, to the existing SQL queries. All of the delete statements belonging to one cascade structure are added to a batch and executed at the end. It uses single connection for each parent/child(ren) tree. Every cascade structure is a logical group.

Manual propagation (cascade) of deletes to child tables

Every time there is a change in the relationship between tables, this process must be modified to reflect that change. Table relationship changes occur when clients decide to make significant customizations to the application.

Cascade relationships

The developer must manually code the parent/child relationships between tables. For example, in order to delete records for the IM_DOC_HEAD table, records must be deleted from children tables in the following sequence of steps. Note that table sequence is not important within a single step.

Step 1

```
Delete from: IM_DETAIL_MATCH_INVC_HISTORY
Delete from: IM_INVOICE_DETAIL_ALLOWANCE
Delete from: IM_QTY_DISCREPANCY_ROLE
Delete from: IM_QTY_DISCREPANCY_RECEIPT
```

Step 2

```
Delete from: IM_DOC_DETAIL_COMMENTS
Delete from: IM_MANUAL_GROUP_INVOICES
Delete from: IM_DOC_HEAD_COMMENTS
Delete from: IM_INVOICE_DETAIL
Delete from: IM_DOC_HEAD_LOCK
Delete from: IM_FINANCIALS_STAGE
Delete from: IM_COST_DISCREPANCY
Delete from: IM_RESOLUTION_ACTION
Delete from: IM_REVERSAL_RESOLUTION_ACTION
Delete from: IM_SUMMARY_MATCH_INVC_HISTORY
Delete from: IM_QTY_DISCREPANCY
Delete from: IM_DOC_DETAIL_REASON_CODES
Delete from: IM_FINANCIALS_STAGE_ERROR
Delete from: IM_DOC_NON_MERCH
```

Step 3

Delete from: IM_DOC_HEAD

Cascade relationships are wired in the BatchPurge.java.

Assumptions and scheduling notes

Every time there is a change in the relationships among tables, the BatchPurge process has to be updated to accommodate these changes.

Major modules

BatchPurge

This class implements the batch delete process for the ReIM base application.

Primary tables involved

The following list includes the tables on which the purging algorithm is applied:

- IM_DOC_GROUP_LIST
- IM_DOC_HEAD
- IM_PARENT_INVOICE
- IM_REASON_CODES

Other tables of less significance also get purged.

EDI invoice upload

Overview

EDI invoice upload is a standardized file format specification designed for vendors to send invoicing information electronically. The EDI invoice upload batch process performs the following:

- Reads each transaction within the file.
- Runs a file format validation (verifying file descriptors and line numbers; ensuring that numeric fields are all numeric and that character fields are all characters; looking for the invalid ordering of record type—THEAD followed directly by another THEAD; and so on). Certain file formatting errors cause the process to terminate with a message indicating the problem. A limited set of data validation errors cause the invalid transaction to be written to error tables (IM_EDI_REJECT_DOC_XXX) where the data can be corrected through an online process. The rest of the data validation errors cause the invalid transaction to be written to a reject file where a user must correct the problems and re-run the file.
- Validates the data against the ReIM system and the merchandising system (such as RMS).

- Any errors found are recorded in an error log so that users can fix any transactions that were rejected to file.
- Adds the data to the ReIM system. All valid transactions are written to the IM_DOC_xxx, IM_INVOICE_xxx, IM_PARENT_xxx tables.

Assumptions and scheduling notes

- This process must be run before the auto-match process.
- All SKUs must have an associated primary UPC because the EDIUpload File can only receive UPCs.
- All quantities are assumed to be in 'EA'ches when uploaded.

Restart and recovery

If the EDI invoice upload aborts without processing an entire file, the file needs to simply be rerun. When this action is completed, there will be multiple errors for the transactions that were successfully uploaded and the other transactions will be uploaded at that time as well. If the cause of the aborted process is software related, this fix may *not* solve the issue. Other steps may be required to ensure that the process completes its entire initial run.

Primary tables involved

- IM_DOC_HEAD
- IM_INVOICE_DETAIL
- IM_INVOICE_DETAIL_ALLOWANCE
- IM_DOC_NON_MERCH
- IM_DOC_DETAIL_REASON_CODES
- IM_PARENT_INVOICE
- IM_PARENT_INVOICE_DETAIL
- IM_PARENT_NON_MERCH
- IM_EDI_REJECT_DOC_DETAIL
- IM_EDI_REJECT_DOC_DETAIL_ALLOW
- IM_EDI_REJECT_DOC_HEAD
- IM_EDI_REJECT_DOC_NON_MERCH

Receiver adjustment

Overview

To resolve a cost discrepancy, the user can select a 'Receiver Cost Adjustment' action from the cost resolution screen. Similarly, to resolve a quantity discrepancy, the user can select a 'Receiver Unit Adjustment' action from the quantity resolution screen. The actions are written to the IM_RESOLUTION_ACTION table in an unrolled status with the amount of adjustment. The IM_INVOICE_DETAIL table also receives a flag that signifies 'pending adjustment' for the invoice line.

At the same time, the actions are written to the IM_RECEIVER_COST_ADJUST and IM_RECEIVER_QTY_ADJUST tables to indicate the expected receiver adjustment amount on the RMS (or equivalent merchandising system) side. In sum, these two tables serve as the staging tables for the RMS (or equivalent merchandising system) process to actually perform the adjustment. For a receiver cost adjustment, IM_RECEIVER_COST_ADJUST holds the order unit cost for the item after the adjustment. For a receiver unit adjustment, IM_RECEIVER_UNIT_ADJUST holds the received quantity for the item on the shipment after the adjustment.

The process compares the unit cost and/or quantity received for the item on the shipment with the expected unit cost and/or quantity on the IM_RECEIVER_COST_ADJUST and/or IM_RECEIVER_UNIT_ADJUST tables. If a match exists, the receiver cost and/or unit adjustment has occurred in RMS (or the equivalent merchandising system). As a result, the process sets the 'pending adjustment' flag on IM_INVOICE_DETAIL table to false for the invoice line. The reason code actions are only rolled up for an invoice if no invoice lines on the invoice have any pending adjustments.

Because ReIM cannot control when and how the receiver adjustments are happening on the RMS side (or the equivalent merchandising system), records written to the IM_RECEIVER_COST_ADJUST and IM_RECEIVER_UNIT_ADJUST tables are considered final. As a result, when the user resolves a cost or quantity discrepancy, the receiver adjustment must fully resolve a discrepancy before the user leaves the screen, and there should be no re-route actions involved. On the RMS side, the amount of adjustment must be exactly the same as expected.

The IM_PARTIALLY_MATCHED_RECEIPTS table holds the amount of a receipt item that has been matched during invoice matching. The quantity received on the SHIPSKU table subtracts the quantity matched on the IM_PARTIALLY_MATCHED_RECEIPT table, giving the available to match quantity for the receipt item. Auto-match, summary matching, detail matching and quantity discrepancy resolution processes all keep track of the matched quantity bucket to determine how much of the receipt item has already been matched and how much of the receipt item remains available to be matched. In the case of a Receiver Unit Adjustment, the IM_PARTIALLY_MATCHED_RECEIPTS table is updated to reserve the entire remaining unmatched bucket for the receipt item. This logic prevents the adjusted receipt quantity from being used for any other matching or quantity resolutions.

Assumptions and scheduling notes

The receiver adjustment process must be run after a client-written check for Receiver Cost Adjustments (RCAs) and and Receiver Unit Adjustments (RUAs).

Primary tables involved

- IM_COST_DISCREPANCY
- IM_QTY_DISCREPANCY
- IM_RECEIVER_COST_ADJUST
- IM_RECEIVER_UNIT_ADJUST
- IM_RESOLUTION_ACTION
- IM_INVOICE_DETAIL
- IM_DOC_HEAD
- IM_PARTIALLY_MATCHED_RECEIPTS

Auto-match (AutoMatchService)

Overview

Auto-match is a system batch process that attempts to match invoices to receipts without manual intervention. Invoices that are in ready for match, unresolved, or multi-unresolved status are retrieved from the database to be run through the auto-match algorithm.

The three inputs into the auto-match process include the following:

- 1 Invoices
- 2 Receipts
- 3 Purchase orders

ReIM ‘owns’ invoices, while receipts and purchase orders are ‘owned’ by a merchandising system, such as RMS.

The processing consists of two levels: summary and detail. Summary level matching attempts to match all invoices to receipts at a summary level. Detail level matching attempts to match all invoices (that do not match at a summary level) to receipts at a line item level.

The auto-match process attempts to match the invoices to receipts to the best of its abilities. The process assign different statuses according to the level of matching achieved.

If an invoice arrives prior to a receipt (for a particular PO), the auto-match process attempts only to match invoice unit cost to PO unit cost.

When a complete match cannot be made, manual intervention is required through online processes.

The four algorithms

The following four algorithms comprise the auto-match process:

- 1 **Cost pre-matching**
This process identifies any cost discrepancies prior to the arrival of receipts. If no receipts exist for the PO location, the invoices are sent to the cost pre-matching algorithm. Cost pre-matching is where unit costs on the invoice are compared with unit costs on the purchase order at a line level. If a match can be obtained, the invoice remains in ready-for-match status and is retrieved again for matching once the receipt comes in. If no match can be obtained, a cost discrepancy is created and routed immediately.
- 2 **Summary matching**
Invoices are grouped with receipts based upon purchase order location. A match is attempted for all invoices and receipts for the PO location. The invoices' total extended costs are summed and compared with the receipts' total extended costs. Based on a supplier option, the invoices' total quantity is summed and compared with the receipts' summed total quantity. If a match is achieved, all invoices and receipts are set to matched status. Otherwise, one-to-one matching is attempted for the PO location.
- 3 **One-to-one invoice matching**
This processing attempts to match a single invoice to a single receipt for the applicable PO location. If all invoices and receipts are set to matched status, the next PO location is processed.

If a multi-unresolved scenario exists (where more than one invoice can be matched with one or more receipts), all un-matched invoices are given the multi-unresolved status and no further processing occurs for this PO location.
- 4 **Detail matching**
During detail matching processing, an attempt is made to match each line on the invoice to an unmatched receipt line for the same item. Both the unit cost and quantity are always compared at the line level. If both the cost and quantity match, the invoice line and receipt line are placed into matched status. If the cost fails or the quantity fails, the cost or quantity discrepancies are generated and routed.

Assumptions and scheduling notes

- Auto-match must *not* be run during the day when there are users online interacting with the system.
- Both the invoice unit cost and the PO's unit cost must be expressed in the same currency. In order to compare the invoice unit costs with the purchase order's unit costs, auto-match does not engage in currency conversion.

The system assumes that tolerance costs are always in the system's primary currency. If RMS is the applicable merchandising system, auto-match performs currency conversion if the currency on the order is different from the primary currency. RMS's existing currency conversion engine is used to perform this conversion. If RMS is not being utilized, another currency conversion engine must be provided to support this functionality.

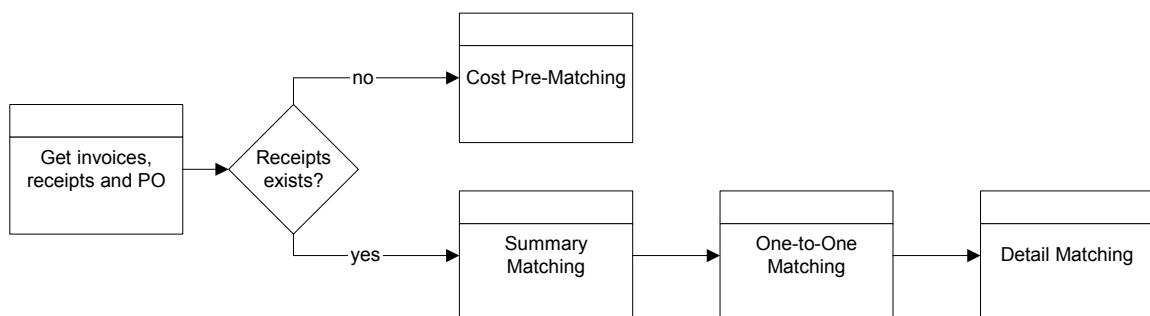
- The quantities on the invoice must be expressed in the same unit of measure as the quantities on the receipt. Auto-match performs no unit of measure conversion.
- The batch process runs after EDI upload (Invoice Matching) and Receipt upload (Merchandising system, such as RMS).

Post processing

- Auto-match automatically invokes the 'best terms calculation' for invoices that it matches.
- Auto-match automatically post invoices that it matches.

High-level flow diagram

The following diagram offers a high-level view of the processing logic utilized within the auto-match batch process.



ReIM's auto-match flow

Primary tables involved

- IM_DOC_HEAD
- IM_INVOICE_DETAIL
- SHIPMENT (RMS)
- SHIPSKU (RMS)
- IM_PARTIALLY_MATCHED_RECEIPTS
- ORDHEAD (RMS)
- ORDSKU (RMS)
- ORDLOC (RMS)
- IM_TOLERANCE_DEPT
- IM_TOLERANCE_SUPP
- IM_TOLERANCE_SYSTEM
- IM_COST_DISCREPANCY
- IM_QTY_DISCREPANCY
- IM_QTY_DISCREPANCY_RECEIPT
- IM_QTY_DISCREPANCY_ROLE
- IM_SUPPLIER_OPTIONS
- IM_SYSTEM_OPTIONS

Reason code action rollup

Overview

Reason code actions are resolutions assigned at the discrepancy line level. A number of fixed actions are available to resolve a line item discrepancy; the specific results depend on the action.

The resolution posting process sweeps the IM_RESOLUTION_ACTION table and creates debit and credit memos as needed. Only a single debit or credit memo is created per invoice, with line details from all related actions.

This process does *not* delete these records when completed; rather, they are deleted after posting.

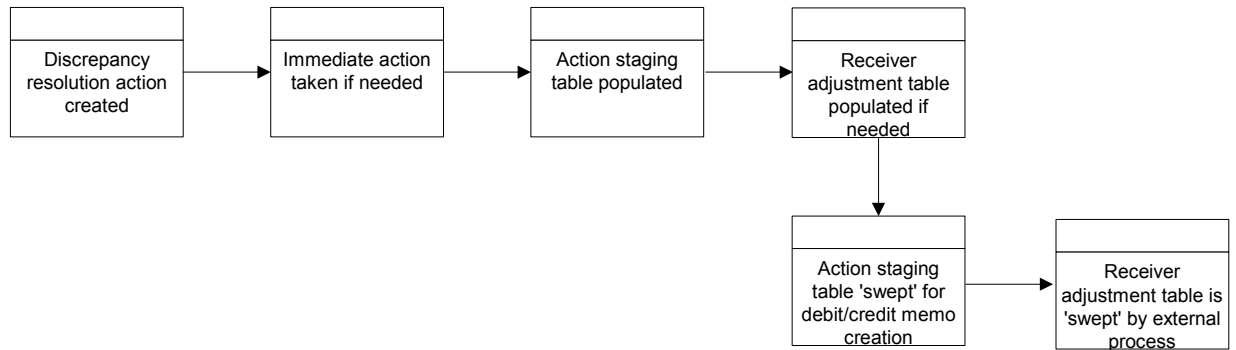
A separate, client-created batch process sweeps the receiver adjustment table. The action staging table is used during posting to post the reason code actions to the financial staging table.

Assumptions and scheduling notes

- If the receiver adjustment table sweep does not occur before the auto-match process, there will be a delay of one day before the detail matching work can be performed for the applicable invoice.
- The memo staging table sweep must occur before the posting batch process, or a delay of one day results before posting can occur.

High-level flow diagram

The following diagram offers a high-level view of the processing logic utilized within the reason code action rollup batch process.



ReIM's reason code action rollup flow

Primary tables involved

- IM_DOC_HEAD
- IM_INVOICE_DETAIL
- IM_PARTIALLY_MATCHED_RECEIPTS
- IM_RESOLUTION_ACTION
- IM_RECEIVER_COST_ADJUST
- IM_RECEIVER_UNIT_ADJUST

Disputed credit memo action rollup

Overview

When a disputed credit memo is first created as a reversal to a debit memo, cost or quantity discrepancies are generated for each line on the credit memo, and the original debit memo reason codes are associated with the new credit memo detail lines.

As the user takes actions to resolve the discrepancy online, a record is written to the IM_REVERSAL_RESOLUTION_ACTION table for each resolution action taken. The only actions allowed to resolve the discrepancy are Deny Dispute or Approve Credit in Disputed Status. However, the user can choose multiple reason codes associated with Deny or Approve actions to resolve the disputed line. Also, the user can either resolve the disputed line completely, or partially resolve it. Upon complete resolution of a disputed line, the cost or quantity discrepancy is deleted from the system.

The disputed credit memo action rollup process checks the records on the IM_REVERSAL_RESOLUTION_ACTION table and rolls up the credit memo detail lines by document/item/reason code. The rollup occurs only if all lines on a disputed credit memo have been completely resolved (that is, no cost or quantity discrepancy records remain for the credit memo).

After the rollup, a new set of detail lines associated with the resolution reason codes replace the original set of detail lines associated with the debit reason codes on the IM_DOC_DETAIL_REASON_CODES table. The new credit memo lines are in Approved or Denied status depending on the resolution action. The credit memo header status is updated to Approved status. The lines that are approved are rolled up to calculate the header level total cost and total quantity. Non-merchandise costs can be associated with a credit memo that is created as a debit memo reversal, but no resolution actions can be taken on non-merchandise costs. Non-merchandise costs should be included in the credit memo's total cost.

Assumptions and scheduling notes

The disputed credit memo action rollup must occur before resolution posting and after receiver adjustment.

Primary tables involved

The following tables are used for the debit memo reversal, resolution, and rollup processes:

- **IM_DOC_HEAD**
This table holds the document header information.
- **IM_DOC_DETAIL_REASON_CODES**
This table holds the document detail information by item/reason code. Before resolution rollup, this table holds the document detail information based on the original debit reason codes. After resolution rollup, this table holds the document detail information based on the reason codes used to resolve the disputed credit memo lines.
- **IM_REVERSAL_RESOLUTION_ACTION**
This table holds the resolution actions the user takes to approve or deny the disputed credit memo line.
- **IM_COST_DISCREPANCY**
This table holds the disputed credit memo lines for a debit memo cost reversal.
- **IM_QTY_DISCREPANCY**
This table holds the disputed credit memo lines for a debit memo quantity reversal.
- **IM_QTY_DISCREPANCY_ROLE**
This table holds the routing information for a credit memo quantity.

Resolution posting action rollup

Overview

For each invoice, the batch process engages in the following high-level steps:

- 1 Performs any resolution actions (for example, instigates the creation of payment documents).
- 2 Calls the posting process to write applicable financial accounting transactions to the financials staging table, IM_FINANCIALS_STAGE.

The processing occurs after discrepancies for documents have been resolved by resolution documents. Once all of the resolution documents for a matched invoice are built, and all of the RCA/RUA external processing has been confirmed, the process inserts financial accounting transactions to the financials staging table, to represent the resolution and consequent posting of the invoice. The process also inserts financial accounting transactions for the approved documents that are being handled.

Once all of the transactions have been written, the process switches the status of the current invoices/documents to 'Posted', and then moves on to the next invoice/document.

If a segment look-up fails, the failed record is written to a financials error table.

Assumptions and scheduling notes

Before posting can occur, the following information must be set up:

- Set up segment definitions in the system.properties.
- Define GL account segments on the GL Options screen.
- Specify all the accounts using the GL Cross Reference screen.

The dynamic segments for a GL account can be any or all of the following designated segments:

- Country
- Location
- Dept
- Class

If dynamic segments are defined, the values for the segments must be defined in the applicable tables, IM_DYNAMIC_SEGMENT_DEPT_CLASS or IM_DYNAMIC_SEGMENT_LOC.

Primary tables involved

- IM_DOC_HEAD
Holds the matched and approved documents.
- IM_DOC_NON_MERCH
Holds the non-merchandise costs for invoices.

Lookup tables that must be populated

- IM_GL_OPTIONS
Order of segments and dynamic segments defined.
- IM_GL_CROSS_REF
Account values defined for account types and account codes.
- IM_DYNAMIC_SEGMENT_DEPT_CLASS
Accounts defined for each department/class combination.
- IM_DYNAMIC_SEGMENT_LOC
Accounts defined for each location/company combination.

Table to which the process posts data

- IM_FINANCIALS_STAGE
 - Transaction code
 - Debit/credit indicator
 - Invoice ID
 - Invoice date
 - Supplier
 - Purchase order (if available)
 - Shipment/receipt (only if an unmatched receipt record is being written)
 - Currency
 - Amount
 - Best terms ID
 - Terms date
 - Pre-paid indicator
 - Comments
 - Create user ID
 - Create date-time
 - Segments that determine the mapping account in the external financial system (as defined in the IM_GL_CROSS_REF table).

EDI invoice download

Overview

The EDI invoice download process retrieves debit memos, credit note requests and credit memos in 'approved' or 'posted' status from the resolution posting process and creates a flat file. The client converts the flat file into an EDI format by the client and sends it via the EDI invoice download transaction set to the respective vendors.

Assumptions and scheduling notes

- All data is valid in the IM_DOC_HEAD tables. ReIM does not validate details.
- Auto-match must run prior to the EDI invoice download.

Primary tables involved

The EDI invoice download batch process reads from the following tables:

- IM_DOC_HEAD
- IM_DOC_DETAIL_REASON_CODES
- IM_DOC_NON_MERCH
- IM_DOC_DETAIL_COMMENTS

Restart and recovery

If the EDI invoice download aborts while processing, an incomplete file is generated. To generate a complete file, the process simply needs to be rerun and allowed to fully process. If the cause of the aborted process is software related, this action might *not* solve the issue; other steps may be required to ensure that the process completes its entire initial run.