

User and Developer Guide



User and Developer Guide



This is version 7.3.0.1 of the KonaKart User Guide

This User Guide can be downloaded in PDF format from http://www.konakart.com/docs/KonaKart_User_Guide.pdf
[KonaKart_User_Guide.pdf]

Legal Notices

(c) 2006 DS Data Systems UK Ltd, All rights reserved.

DS Data Systems and KonaKart and their respective logos, are trademarks of DS Data Systems UK Ltd. All rights reserved.

The information in this document is free; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This documentation is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

Table of Contents

1. Introduction	1
What is KonaKart ?	1
Who is the software intended for?	1
Retailer	1
Solution Provider / System Integrator / OEM	1
ISP	1
Important changes from v.6.5.0.0	2
2. KonaKart Information	3
Community and Enterprise Versions	3
Is KonaKart Open Source?	5
Is the full source code available?	5
3. KonaKart Features	7
General Functionality	7
JSR 286 Liferay Portlet	7
Content Management System Integration	7
Setup/Installation	7
Design/Layout	7
Multi-Store	7
Multi-Vendor	8
Customer Functionality	8
Customer Groups - Wholesale/Retail	9
Call Center Functionality	9
One page checkout	9
Checkout without registration	9
Products	9
Digital Downloads	10
Bookable Products	10
Product Bundles	10
Gift Certificates	11
Reward Points	11
Indexed Search	11
Suggested Search	11
Product Tags and Tag Groups (Faceted Search)	11
Merchandising	11
Promotions	12
Marketing - Customer Tags and Expressions	13
Advanced Search Engine Optimization (SEO)	13
Reporting	13
Payment Functionality	13
Recurring Billing	13
Shipping Functionality	14
Tax Functionality	14
Returns	14
Refunds	14
PDF Invoices	14
Java Message Queue Integration	14
Customer Events	15
4. Architecture	16
Software Architecture	16
Deployment Architecture	18
5. Installation	19

Before You Begin	19
Platforms Supported	19
Pre-requisites	19
Install Java	19
Create a Database	19
Upgrading the Database between releases of KonaKart	20
Install KonaKart	20
Installing KonaKart on Windows	20
Installing KonaKart on Unix/Linux	20
Silent Mode Installations	21
Graphical Installation Wizard	22
Manual Installation	34
Starting Up and Shutting Down KonaKart	36
Starting up KonaKart	36
Shutting down KonaKart	37
Setting up KonaKart as a Windows Service	37
Default Admin App Credentials	38
Admin Password Validation	38
Super User	39
Installation Notes for Databases	39
Defining the Database Parameters	39
Encrypting the Database Parameters	40
Defining the Database Parameters - Using JNDI	41
Notes for DB2 and Oracle	42
Notes for Postgresql	42
Notes for MySQL	43
Notes for Microsoft SQL Server	43
6. Installation of KonaKart Enterprise Extensions	44
Before You Begin	44
Licensing	44
Pre-requisites	44
Create a Database	44
Installing Enterprise Extensions	45
Installing KonaKart Enterprise Extensions on Windows	45
Installing KonaKart Enterprise Extensions on Unix/Linux	45
Silent Mode Installations	46
Graphical Installation Wizard	47
Manual Installation of the Enterprise Extensions	59
Users created by the Enterprise Extensions Installation	66
Single Store Mode	67
Multi-Store Multiple DB Mode	67
Multi-Store Single DB Mode with Shared Customers	67
Multi-Store Single DB Mode NON-Shared Customers	68
7. Installation of KonaKart on other Application Servers	69
General Notes on Installing KonaKart on Application Servers	70
Edit Config Files - Admin Application Functionality	70
Email Properties File	72
Reporting Port Numbers and Report Location	72
Configuring Parameters for Images	72
Setting the Optimum Memory Values	73
Installing KonaKart on BEA's WebLogic Application Server	73
Installation	73
Configuration	74
Installing KonaKart on JBoss	74

Installation	74
Configuration	75
Installing KonaKart on IBM's WebSphere Application Server	76
Installation	76
Configuration	76
Installing KonaKart on IBM's WebSphere Application Server Community Edition	76
Installation	77
Configuration	77
Installing KonaKart on GlassFish	77
Installation	77
Configuration	77
Installing KonaKart on JOnAS with Tomcat	77
Installation	78
Configuration	78
Installing KonaKart on JOnAS with Jetty	78
Installation	78
Configuration	79
8. Upgrading	80
Recommended Upgrade Steps	80
Backup	80
Installation of the new Version	80
Database Upgrade	81
Test new Installation against Upgraded Database	81
Re-build Custom Code	81
Upgrade the Struts Storefront	81
Assess Changes to WebApp Configuration files	82
Assess Changes to Tomcat Configuration files	82
KonaKart Directory Structure Changes	82
KonaKart Message Changes	82
Documentation and javadoc Changes	83
Admin App Changes	83
9. Administration and Configuration	84
KonaKart Administration Application	84
Main Features	85
Reporting	87
Reporting - BIRT Viewer Security	87
Role-based Security and Configuration	88
File-based Configuration	89
Launching the Admin App	92
Configuring KonaKart for HTTPS / SSL	92
Editing the KonaKart Configuration Files	92
Changing the Editable File List in the Admin App	93
KonaKart Properties Files	94
Configuration of Messages	95
Switching to Database Messages	95
KKMessages Utility	95
Logging	96
Internationalization of KonaKart	97
Translating the KonaKart Application	97
Translating the country names	99
Translating the KonaKart Admin Application	99
Changing the Logo on the KonaKart Admin Application	100
Changing the Date Format in the KonaKart Application	101
Formatting of Addresses	101

Email Configuration	102
Modifying the Email Templates	103
Adding Custom Business Objects for use in Velocity Templates	104
Search Engine Optimization (SEO) Features	104
Sitemap Generation	105
Caching	105
Adding Custom Functionality to the Admin App	106
Adding Panels	107
Adding Custom Configuration Panel	108
Adding Buttons	108
Adding A Custom Application - Insert Product Wizard	108
Searching with wildcards	109
Case In-Sensitive Searching	111
Making something happen when a product needs to be reordered	111
Making something happen when the state of an order changes	112
PDF Invoices	114
Activating a Promotion	115
Applying Promotions to Products	116
Displaying Coupon Entry Fields in your Store	117
Configuring Digital Downloads	117
Configuring Bookable Products	118
Import/Export of KonaCart Data	118
Export of Orders using exportOrder	118
Import/Export of KonaCart Data using XML_IO	119
Custom Imports Using the Importer Panel	122
Reset Database Tool	124
Multiple Prices for Products	127
Sale Price	128
Tier Pricing	129
Dynamic product prices	131
Tax Configuration	131
Tax algorithm and numeric precision	132
Validation of Order Totals	133
Multiple Quantities for Products	133
Default Sort Order for Products	135
Bundle Configuration	135
Product Options	136
Product Tags	137
Managing Product Reviews	138
Using CLOBs for Product Descriptions	138
Credit Card Refunds	138
Saving and Editing of Credit Card details	139
Configuration of Admin Application	139
Configuration of Store Front Application	140
Edit Order Number and Custom Fields	141
Shippers and Shipments	142
Wish Lists	143
Gift Registries	143
Gift Certificates	143
Enable Gift Certificates	144
Creating a Gift Certificate	144
Creating a new Admin App User	147
Creating New Roles	148
Default Customer Configuration	148

Making Customers Invisible	148
Customer Groups	149
Auditing	150
Custom Credential Checking	150
Custom Credential Checking - LDAP	151
Multi-Store Configuration and Administration	152
Introduction	152
Configuring KonaKart to function in Multi-Store Mode	152
Multi-Store Configuration	155
Product Synchronization	161
Scheduling in KonaKart	163
Configuring Quartz to execute KonaKart jobs	163
Customizing the KonaKart jobs	167
Deletion of Expired Data	168
Data Integrity	168
Executing the Data Integrity Checker from a Script	169
Configuring KonaKart to use Analytics Tools	170
Configuring KonaKart to use Google Analytics	170
Configuring KonaKart to use Other Analytics Tools	172
Setting up RMI Services	172
Step by Step Guide to setting Up KonaKart to use RMI	172
Integrating a Java Message Queue	175
Setting Up The Java Message Queue	176
Monitoring The Java Message Queue	178
Changing the standard password encryption algorithm	178
10. Marketing - Customer Tags and Expressions	179
What are Customer Tags?	179
What are Expressions?	180
Tutorial for creating an expression using the standard customer tags	180
How to set Customer Tag Values in Java code	184
11. Reward Points	186
Configuration of Reward Points	186
Technical Details	190
12. Solr Search Engine	191
Configuring KonaKart to use Solr	191
Instructions	191
Customization of Solr	191
Forcing Solr Usage	192
Suggested Search	193
Suggested Search using Solr terms	193
Suggested Spelling	194
Functionality	194
Setup	194
Faceted Searching	196
Prices, Manufacturers and Categories	196
Custom Faceted Search	197
Multi-Valued Facets	203
13. Payment, Shipping and OrderTotal Modules	207
Module Types	207
Payment Modules	207
Shipping Modules	208
Order Total Modules	210
How to Create a Payment Module	210
Introduction	211

Study the "KonaPay" APIs	211
Choose which Interface Type you want for your users	211
Sign up for a Test Account with "KonaPay"	212
Determine which of the existing payment modules is the closest match	212
Copy the files of the closest match as the starting point	212
Define the configuration parameters	212
Understanding the Configuration Options	214
Add the "KonaPay" gateway to the Admin App	215
Implement the PaymentInterface	215
NameValuePair[] Parameters	216
Implement the Action code	216
Save IPN details	216
Save the gateway response to a file	216
Send payment confirmation email	217
Struts mapping	217
Build, Deploy and Test	218
How to Create a Promotion Module	219
Determine which of the existing Promotion modules is the closest match	219
Copy the files of the closest match as the starting point	219
Define the configuration parameters	219
Add the new promotion module to the Admin App	221
Build, Deploy and Test	221
14. Recurring Billing	222
Payment Schedule	222
Subscription	222
Using a Payment Gateway that supports Recurring Billing	222
Insert a subscription	223
Update a subscription	223
Read the status of a subscription	224
Manual Subscription Management	224
Managing Recurring Billing through KonaCart	224
15. Multi-Vendor Functionality	226
Configuration	226
Setting up a Main Store	226
Setting up a Vendor Store	227
Orders	229
Customer View of the Order	229
Vendor View of the Order	229
Administrator view of the Order	230
Order State Change	231
Order Payment	231
16. Custom Validation	232
Custom Validation for the Store-Front	232
Configuring validation on data entered through the UI	232
Custom Validation for the Admin Application	232
CustomValidation.properties file	232
Fields Supported by Custom Validation	233
CustomValidation Using a Custom Engine	233
17. Custom Product Attributes and Miscellaneous Items	235
Using Custom Product Attributes	235
What types of Custom Attributes can be added to a product?	235
Creating a Custom Attribute Definition and adding it to a Template	236
Entering Template based Custom Attribute data for a Product	237
Displaying the custom data in the storefront Application	237

Miscellaneous Items	237
Product Description Custom Fields	237
18. Programming Guide	239
Using the Java APIs	239
Using the SOAP Web Service APIs	240
Enable the SOAP Web Services	241
Securing the SOAP Web Services	241
Step-by-step guide to using the SOAP APIs:	242
Using the RMI APIs	244
Using the JSON APIs	248
Running Your Own SQL	254
Database Layer Changes from v7.2.0.0	254
Table and Column Name Changes	255
KonaCart/Torque Programming Changes	255
Customizable Source Code	256
Source Code Location	256
Building the Customizable Source	257
Developing the storefront in Eclipse	261
Developing the Product Creation Wizard in Eclipse	261
Customization of the KonaCart Engines	262
KonaCart Customization Framework	262
Adding a New API call	263
Modifying an Existing API call	270
Enabling Engine Customizations	274
Pluggable Managers	275
Adding a Shopping Cart via SOAP	276
How To Add a KonaCart Shopping Cart?	276
Why loosely-coupled?	277
Movie Review Example	277
SOAP client code generation	281
Example Source Code	282
19. Reporting	283
KonaCart Reporting from the Admin App	283
Modifying the Reports	283
Adding New Reports	283
Defining a Chart to appear on the Status Page of the Admin App	284
Reports Configuration	284
Defining the Set of Reports Shown in the Admin App	284
Accessing the Database in the Reports	285
Customer Events	285
Creating Customer Events	286
20. Liferay Portal Integration	287
Introduction	287
Creation of portlet WAR files	287
Installation Instructions	289
Liferay	289
Liferay for the KonaCart Admin Application	290
21. TaxCloud Integration	292
Introduction	292
How does the integration work?	292
Reconciliation	293
Points to Note	293
22. Thomson Reuters Integration	295
Introduction	295

How does the integration work?	295
Auditing of Order Transactions	297
Points to Note	297
23. PunchOut	298
Introduction	298
Customization of the PunchOut message	299
Application PunchOut Code	300

Chapter 1. Introduction

What is KonaKart ?

KonaKart is software that implements an enterprise java eCommerce / shopping cart system. It's main components are:

- A shop application used by customers to buy your products. There are actually two storefront applications; one of which is specific for mobile devices.
- An Administration application to enable you to manage your store.
- Many Customization and Extension features - allowing you to customize and extend the way KonaKart works.

Who is the software intended for?

Retailer

If you are a retailer and looking for a product to develop an on line store, then KonaKart could be a good match. Regardless of your size, KonaKart will provide a powerful solution that should cover most, if not all of your requirements, delivering unparalleled price / performance.

Although KonaKart is very easy to install and to get up and running, it really does require some JSP / Java development and deployment knowledge in order to realistically take a store into production. Therefore, if you have no Java knowledge in your company and you don't intend on using external professional services, then it's probably not the product that you should be using.

If your company has Java competency then you should feel right at home using KonaKart and soon be in a position where you can install and customize the software to cover all of your business needs and integrate with your other systems.

We provide Professional Services and Support Contracts to assist you during the development stage and to ensure that your store continues to run smoothly once in production.

Solution Provider / System Integrator / OEM

KonaKart offers an enterprise level eCommerce solution that you can easily customize to match the requirements of your customers. Most of the customizable areas such as the store front application, payment, shipping and promotion modules are open source. Also, the KonaKart engine implements a documented API, on top of which you can write integration modules and custom features in order to personalize your KonaKart offering.

We offer a Partner Program, Professional Services and Support Contracts to help you be successful and profitable in your eCommerce projects.

ISP

KonaKart is a very good match for ISPs offering Java hosting and software solutions. You may offer the community edition completely free of charge to your customers, with point and click installation to easily enable them to create their on line store.

The enterprise version of KonaKart which includes multi-store, is a good solution for providing many stores in a resource efficient manner.

Important changes from v.6.5.0.0

In version 6.5.0.0 of KonaKart a new storefront application was introduced with a more modern and appealing design. The new storefront uses Apache Struts 2 (rather than Struts 1) as the Model View Controller. It makes use of JQuery and Ajax to provide a more productive interface with less screen refreshes and it no longer uses GWT for One Page Checkout functionality and for the Suggested Search widget.

The standard version 6.5.0.0 installer installs the new Struts 2 based storefront.

Chapter 2. KonaKart Information

Community and Enterprise Versions

Since version 3.2.0.0, KonaKart comes in two separate installations:

- A free Community Edition which can be downloaded from our web site downloads page.
- Enterprise Extensions which we charge for. See our web site prices page for pricing details.

The Community Edition is generally intended for small businesses. A condition of the license agreement is to display "Powered By KonaKart" with a link to our web site, on the main page of the on line store.

The Enterprise Extensions are available as a separate installation kit which is installed on top of the community edition to provide more features and functionality. Although we would prefer you to keep it, the "Powered by KonaKart" link is not mandatory for a KonaKart based store when the Enterprise Extensions are installed. Currently the features present only in the Enterprise Extensions are:

- Storefront application for mobile devices. The storefront has been designed for customers using mobile phones and tablet computers. It allows you to access all of the functionality using touch rather than by use of the mouse.
- KonaKart Client Engine source code. It includes the full source code of the client engine as well as a utility for creating an Eclipse project for customizing the storefront application. Both versions of KonaKart include the source code for the JSPs and Struts action classes but the EE also includes the client engine so that all aspects of the storefront can be easily customized. Note that even extensive storefront customizations remain compatible with future versions of KonaKart since they communicate with the KonaKart eCommerce engine through the APIs, which remain backwards compatible.
- Multi-Store. This mode allows you to run an unlimited number of stores with a single KonaKart installation and a single database schema. Even when deploying a single production store, it's useful to run KonaKart in multi-store shared products and categories mode in order to create stores for managing products in staging environments.
- Multi-Vendor. KonaKart allows vendors to manage their own products and orders through the Admin App. The storefront application displays products from all vendors and allows the customer to checkout with any selection of products in a single order.
- Indexed Search. Indexed search using Lucene search technology (Solr) gives you a lightning fast search experience even for very large product catalogs as well as powerful faceted searches. Digital download products (.txt and .pdf) can be indexed in the Solr search engine and text fragments (snippets) surrounding search keywords can be returned.
- Suggested Search. As you type into the search box, a list of suggested search items appear matching the typed letters. The suggestions are weighted by popularity so the most common suggestions are shown first.
- Spelling suggestions. If a search provides no results, a list of suggested words that exist in the catalog are presented to the customer.
- Advanced marketing functionality that allows you to capture customer data as the customer uses your KonaKart eCommerce store; and to use that data within complex expressions in order to show dynamic content, activate promotions and send eMail communications. For example, you could show a banner

or activate a promotion only to customers in a certain age bracket that have Product A in their wish list and at least \$50 worth of goods in their cart.

- Promotion evaluation directly for products, rather than as Order Total modules. This allows a customer to view the available promotions for a product without having to add it to the cart.
- Configurable product options where a customer may enter the quantity or price of an option as well as text to customize a product.
- Unlimited number of custom product attributes. Each attribute may include metadata for validation and widget selection during data entry using the Admin App.
- Unlimited number of miscellaneous objects may be associated with products and categories.
- Wish List functionality. Registered customers can add products to a wish list. The KonaKart API supports multiple named wish lists for each customer.
- Gift Registry functionality. Registered customers can create a gift registry which can be made public or private. Public gift registries can be searched for by shoppers and items within the registry can be bought and shipped directly to the address of the registry owner. The store front application contains an implementation of a wedding registry.
- Reward Points which enable you to increase customer loyalty and increase sales by rewarding customers for purchases as well as other actions such as registering, writing a review, referrals etc. The points may be redeemed during checkout.
- Gift Certificates. Gift Certificate products may be connected to any type of promotion and activated through a coupon code contained within the certificate.
- The KonaKart APIs are available via Java RMI (Remote Method Invocation), JSON (application engine only with JSON) and JavaScript. The Community Version of KonaKart allows you to call the APIs as Java methods and through SOAP.
- jQuery plugin. This plugin allows you to easily call the KonaKart JSON APIs directly from JavaScript.
- Shopping Widgets. These are widgets that can be introduced in any web page using a few lines of JavaScript. They allow you to promote your products from web sites such as social networks (Facebook etc.) and blogs.
- Job Scheduling. This is achieved with an integration of the Open Source Quartz scheduler. There is a framework for adding your own batch jobs and the source code of some useful example jobs.
- Support for Recurring Billing. Payment Schedule and Subscription objects have been introduced to support recurring billing natively using a KonaKart batch or through a payment gateway that manages the billing process at regular intervals.
- Support for Refunds. Real time credit card refunds through the payment gateway can be managed from the Admin App.
- Support for Shippers and Shipments. Multiple shipments for a single order can be added from the admin app.
- Google Base integration which allows you to publish your product information for inclusion in Google search results.
- Product Synchronization feature to allow the synchronization of products between pre-production and production environments. This feature is available when in multi-store, shared product and shared category mode.

- XML Import/Export feature for KonaKart objects such as product, customer, order etc. This feature can be run with a script, providing arguments to define which objects to import or export. Some example scripts are provided.
- Java Message Queue Integration (Apache ActiveMQ) to support the guaranteed delivery of messages to external systems.
- Digital download products (txt and .pdf) may now be indexed in the Solr search engine and text fragments (snippets) surrounding search keywords can be returned from the search.
- The products within a single store may contain an unlimited number of price and quantity in stock definitions. The price and quantity used, is determined by a catalogId which is passed to the relevant API call. The selection of which catalogId to use can be made using custom business rules.
- The language of the admin app may be changed dynamically.
- PDF invoices can be created and sent to customers as email attachments and downloaded from the storefront application. The created PDF invoices can be stored on disk for archiving purposes or created dynamically whenever they are required.
- Customer events that may be used for reporting purposes. You can log any event such as when a product is viewed or removed from the cart or when a customer starts and completes the checkout process.
- Bookable Products such as courses and tickets may be defined. Each product can have an associated schedule and a list of bookings.
- LDAP module to connect to an LDAP directory in order to validate customer and admin user credentials.
- Flexible configuration of product image loading and scaling.
- Ehcache used to improve performance by caching of data.
- PunchOut functionality allowing an e-procurement system such as SAP to order from a KonaKart store.
- XML sitemap generation to inform search engines about pages that are available for crawling.

Support Packages and Professional Services are available for *all* versions of KonaKart.

Is KonaKart Open Source?

The full source code of the storefront application including the KonaKart client engine, the Struts action classes, the JSPs, the payment modules, order total modules and shipping modules are included in the Enterprise Edition of KonaKart. The Community Edition includes all of the above except the source of the client engine. The Community Edition source code is shipped under the GNU Lesser General Public License.

Is the full source code available?

Under certain circumstances DS Data Systems will sell the entire KonaKart source code, although we prefer to sell an Escrow service where possible.

The Enterprise version of KonaKart contains the complete source code of the Client Engine as well as the JSPs and Struts action classes. This gives you full control over the functionality and look and feel of the storefront application.

For the cases where functionality is missing, and it makes sense for this functionality to reside in the KonaKart eCommerce engines, we normally provide fixed price quotes for implementing the missing features and providing an API which we maintain for new releases. This allows you to easily upgrade when new releases become available and also allows you to be supported. The disadvantage of taking the source code and editing it, is that in most cases this results in a branching of the KonaKart code base which becomes difficult to upgrade and maintain.

Chapter 3. KonaKart Features

General Functionality

- KonaKart supports most popular databases through JDBC. (e.g. MySQL, PostgreSQL, Oracle, DB2, MS SQL Server are all supported in the download package).
- Written in Java. Needs a servlet engine such as Apache Tomcat to run.
- Modular approach with APIs at various levels. The APIs are available as Java APIs, SOAP, JSON and RMI. The JSON APIs are used by the jQuery plugin that allows the KonaKart application engine to be called using AJAX JavaScript calls. The variety of protocols supported, promote connectivity even from outside of the company firewall and allow client side applications (i.e. .Net, MS Excel etc.) to use the KonaKart engine. They also allow you to easily integrate eCommerce functionality into your current application, which may be for example, a content management system.
- Completely multilingual.
- Many objects contain custom fields to facilitate personalizations

JSR 286 Liferay Portlet

- In order to easily integrate the KonaKart Storefront and Admin applications into the popular Liferay portal environment, we supply ANT tasks that automatically generate portlets from the applications which can then be easily imported into Liferay. The storefront application maybe customized within an Eclipse based project to match you requirements before you generate the portlet.

Content Management System Integration

- KonaKart has been successfully integrated with many CMS systems using the APIs. An OpenCms module is available for KonaKart in order to demonstrate how to integrate KonaKart functionality within this popular Open Source content management system using the SOAP APIs.

Setup/Installation

- Simple click and run installation through an installer.

Design/Layout

- The store front application uses a JSP / Struts design. The source code of the JSPs and Struts Action classes is provided in the download so that they may be customized.
- It is relatively easy to write a UI in the technology of your choice by directly calling the KonaKart APIs. One example is the jQuery demo which shows how the KonaKart Application Engine API may be easily accessed using AJAX JavaScript calls made available by the KonaKart jQuery plugin.

Multi-Store

- KonaKart provides advanced multi-store functionality to enable you to run your stores from a single KonaKart deployment and a single database.
- Shared customers mode allows you to share customers between all stores. This is very useful for shopping mall applications since a customer only has to register once in order to shop in many different stores.

- Shared products mode allows you to share products between stores. This means that the products may only exist once in the database for maintenance purposes, but may be included in many stores. The product price may be different for each store in which the product is included. This is a useful mode for companies setting up stores in different countries, selling the same products.
- Shared products and shared categories mode allows you to share products and categories between stores. This mode can be very useful for using stores to provide one or more pre-production environments. KonaKart includes functionality that allows you to synchronize products between stores when working in this mode. Once the products in the pre-production store have been approved, they can be copied to the production store on the click of a button or through the use of a scheduled batch program.
- When KonaKart is configured to be in multi-store mode with a shared database (Engine Mode 2), product searches can span more than one store. They can span all of the stores, or a list of stores can be supplied in the search request.

The ProductSearch object has a searchAllStores boolean which should be set to search all stores. If the search is only for a limited number of stores, then an array of storeId Strings can be set (storesToSearch) to identify which stores should be searched.

- The administration application allows a super user to create a store and then create a store administrator role so that when the store administrator logs into the administration application, he can only administer the store that has been assigned to him.
- For special cases where store data has to be kept within separate DB schemas, KonaKart can be configured to achieve this, although some of the functionality such as the sharing of customers described above, is not available in this mode.
- When in shared customer mode, other data, such as countries, zones and tax information is also shared. This can save a great deal of time in administering a KonaKart multi-store installation as these objects only need to be set up once for all stores.

Multi-Vendor

- KonaKart allows vendors to manage their own products and orders through the Admin App. The store-front application displays products from all vendors and allows the customer to checkout with any selection of products in a single order.

Customer Functionality

- Customers can view their order history and order statuses.
- Customers can maintain their accounts. They have an address book for multiple shipping and billing addresses.
- Permanent shopping carts for guests and registered customers. The permanent cart for guests is managed through cookies.
- Registered and temporary customers can create wish lists.
- Registered customers can create a gift registry which can be made public or private. Public gift registries can be searched for by shoppers and items within the registry can be bought and shipped directly to the address of the registry owner. The store front application contains an implementation of a wedding registry.
- Fast and friendly quick search, advanced search and suggested search features. Search for products by category, by manufacturer or both.

- Product reviews for an interactive shopping experience.
- Number of products in each category can be shown or hidden.
- Global and per-category bestseller lists.
- Dynamic product attributes relationship.
- HTML based product descriptions.
- Display of specials
- Control if out of stock products can still be shown and are available for purchase.
- Customers can subscribe to products to receive related emails/newsletters.
- All emails are template driven and so fully customizable using the Apache Velocity template language.

Customer Groups - Wholesale/Retail

- Control what prices are displayed to customers. i.e. You may show different prices to wholesale customers or company employees etc.
- Enable promotions. Promotions may be enabled only for certain types of customers. i.e. You may want a 3 for 2 promotion to only apply to your retail customers.
- Send communications. You may send out bulk emails only to customers that belong to a particular customer group.

Call Center Functionality

- From the administration application, an administrator can open up a browser displaying the KonaKart eCommerce application and log in on behalf of a customer without requiring the customer's credentials. Once logged in as that customer, he can process orders for the customer and perform all tasks that are normally enabled for the customer when logging in independently.

One page checkout

- One Page Checkout screen using AJAX technology
- Existing customers go directly to the checkout screen where they can add coupons, change shipping methods, payment methods etc. and immediately see the updated total order amount without a screen refresh.
- New customers can add address details as part of the checkout process. They don't have to go through a registration process.

Checkout without registration

- KonaKart may be configured to allow customers to checkout without registering and creating an account.

Products

- Each product may be configured with multiple options. Every configuration may have:

- A unique price
- A unique SKU
- A unique quantity
- A unique available date

Configurable options are also supported where a customer may enter the quantity or price of an option as well as text to customize a product.

- Each product may be associated with a number of images.
- Each product may be associated with up to 4 prices. The actual price displayed can be controlled by the customer group. The Enterprise Extensions package allows you to associate a product with an unlimited number of catalogs, each containing prices and stock information.
- A special sale price with a start and end date may be defined for a product.
- Tier prices or percentage discounts may be defined for volume discounting.
- Each product may contain structured data as well as a description. The structured data can be used for comparing product features.
- Each product may contain an unlimited number of custom attributes. Each attribute may include metadata for validation and widget selection during data entry using the Admin App.
- Each product may contain an unlimited number of miscellaneous items which can have a type as well as a value. i.e. Useful for associating a product with multiple videos and documents.
- KonaKart includes an import/export tool to efficiently load products into the database from a file, and to create a file from the products in the database.

Digital Downloads

- A product may be defined as a digital download.
- When a digital download has been paid for, a download link appears in the customer's private account page.
- The download link can be set to expire after a number of days or after a number of downloads.

Bookable Products

- A product such as a course or ticket may be defined as a bookable product.
- Each bookable product has a start and end date and may have a schedule defining time slots for certain days of the week. Whenever a bookable product is purchased, a booking is created which is associated with the bookable product so that at any point in time, the number of bookings may be monitored.

Product Bundles

- Bundle products can be defined in the Admin App. The quantity of the bundle product is calculated automatically and tools are provided to calculate the cost (optionally using discounts) and the weight.

In the application, the bundled products are available in the product detail screen, the quantity available is calculated and the quantity in stock of the bundled products is decremented automatically when an order is processed.

Gift Certificates

- Gift Certificate products may be connected to any type of promotion and activated through a coupon code contained within the certificate.

Reward Points

- Reward Points enable you to increase customer loyalty and increase sales by rewarding customers for purchases as well as other actions such as registering, writing a review, referrals etc. The points may be redeemed during checkout.

Indexed Search

- KonaKart has been engineered to manage product catalogs containing hundreds of thousands of products. In order to search for these products in an efficient manner, KonaKart may be configured to use the Apache Solr enterprise search server based on the Lucene Java search library.
- This powerful technology not only provides for a lightning fast search experience (including faceted searches), but also incorporates intelligence to correct common problems such as misspellings and plurals which often frustrate shoppers making them go elsewhere. For example, the words Television, Televisions, TV, TVs can all be configured to find televisions.

Suggested Search

- As you type into the search box, a list of suggested search items appear matching the typed letters. The suggestions are weighted by popularity so the most common suggestions are shown first.
- For each product, many suggestion terms are indexed such as the product name, model, category, manufacturer, manufacturer within a category and category for a manufacturer. This greatly increases the usability of the search and directs the user not only to products matching the search string, but also to categories of products or products for a manufacturer or products for a manufacturer within a category etc.

Product Tags and Tag Groups (Faceted Search)

- Tags are attributes that can be associated to a product and can be used to refine product searches.
- The purpose of a tag group is to organize tags and a tag group may be associated to a category so that it can be automatically displayed in a context sensitive fashion when a customer is viewing products belonging to a specific category.

Merchandising

- Display what other customers have ordered with the current product shown.
- For every product you may define a list of products for:
 - Up-selling

- Cross-selling
- Accessories
- Dependent products (e.g. Service Plans or Extended Warranties)

Promotions

- Very powerful promotions sub-system
- All promotions can be associated with one or more coupons. Each coupon can be configured for unlimited use or to be used only for a programmable number of times.
- The following rules can be set for all promotions:
 - Include only some customers (e.g. those that haven't placed an order in the last 60 days) or exclude some customers (e.g. those that have placed an order in the last 60 days). Configure how many times an included customer can use the promotion.
 - Include or exclude customers based on the customer group.
 - Include or exclude customers based on expressions.
 - Include or exclude products based on their category.
 - Include or exclude products based on their manufacturer.
 - Include or exclude any products at a product option granularity. (e.g. Promotion only applies to shoe size 7 or applies to all sizes except size 7)
- Promotions may be cumulative (e.g. A promotion of 10% discount for all hardware products and a promotion of 20% discount for all software products) or promotions may be exclusive, in which case the promotion giving the largest discount is chosen.
- All promotions may be configured to have a start and end date.
- The promotions themselves are KonaCart Order Total Modules which can easily be developed and slotted into the KonaCart architecture. The source code of the available promotion modules is included in the download package.
- The currently available modules are :
 - Order Total Discount
 - The promotion gives a discount on the total amount of the order.
 - You may set a minimum order value, the minum number of products, the minimum quantity of a single product, the discount as an amount or a percentage and whether to apply the discount before or after tax.
 - Product Discount
 - The promotion gives a discount on a product
 - You may set a minimum order value, the minimum quantity of a single product, the discount as an amount or a percentage and whether to apply the discount before or after tax.
 - Buy X get Y free

- If the customer buys X + Y products, he only pays for X.
- Shipping Discount
 - The promotion gives a discount on selected shipping methods.
- Free Product
 - The promotion entitles the customer to receive a free product if the conditions of the promotion are met.

Marketing - Customer Tags and Expressions

- The Enterprise version of KonaKart contains sophisticated marketing functionality that allows you to capture customer data as the customer uses your KonaKart eCommerce store; and to use that data within complex expressions in order to show dynamic content, activate promotions and send eMail communications.
- Later in this document there is a chapter dedicated to this functionality.

Advanced Search Engine Optimization (SEO)

- You have full control over which SEO features to activate. KonaKart allows you to define multi-lingual templates in order to write product information (name, model, manufacturer, category) into:
 - The URL
 - The window title
 - The meta description
 - The meta keywords
- XML sitemap generation. Sitemaps are an easy way for to inform search engines about pages that are available for crawling.

Reporting

- KonaKart is integrated with BIRT which is an open source Eclipse-based reporting system. The default installation package includes a number of useful reports. Others may be easily added through the Admin App or just by copying them to a directory where they are automatically read by KonaKart. Instructions can be found in the Reporting Chapter of this User Guide.

Payment Functionality

- KonaKart implements an API and modular approach for introducing payment gateways. We include the source code of all currently available gateways.
- Disable certain payment services based on a zone basis.

Recurring Billing

- Support for Recurring Billing. Payment Schedule and Subscription objects have been introduced to support recurring billing natively using a KonaKart batch or through a payment gateway that manages the billing process at regular intervals.

- Later in this document there is a chapter dedicated to this functionality.

Shipping Functionality

- Weight, price, and destination based shipping modules
- Free shipping based on amount and destination
- Free shipping on a product by product basis
- Disable certain shipping services based on a zone basis
- KonaCart implements an API and modular approach for introducing custom shipping services. Popular services such as FedEx, UPS and USPS are provided out of the box.
- Shippers and Shipments can be defined and saved in the database. Multiple shipments may be defined for a single order. The shipments (including tracking number) can be included in a notification eMail and are visible from the order details page of the storefront application.

Tax Functionality

- Flexible tax implementation on a state and country basis.
- Set different tax rates for different products.
- Charge tax on shipping on a per shipping service basis.
- Connect with online tax calculation services such as TaxCloud and Avalara.

Returns

- Returns may be managed from the Admin App. KonaCart can support multiple returns per order, each of which may contain different product quantities and have a unique RMA code.

Refunds

- Real time credit card refunds through the payment gateway can be managed from the Admin App. The full details of the refund transactions are saved in the database.

PDF Invoices

- PDF invoices can be created and sent to customers as email attachments and downloaded from the storefront application. The created PDF invoices can be stored on disk for archiving purposes or created dynamically whenever they are required.

Java Message Queue Integration

- To support the guaranteed delivery of messages to external systems the Apache ActiveMQ java message queue has been integrated (Enterprise Extensions only).

A typical example of where this might be useful is where you wish to transfer orders to another system via a message queue once your orders have reached a certain status (eg. once payment has been received).

Customer Events

- Customer Events may be defined to save important event information that may be used for reporting purposes. For example, events may be inserted whenever a product is viewed or removed from the cart or when a customer starts and completes the checkout process.

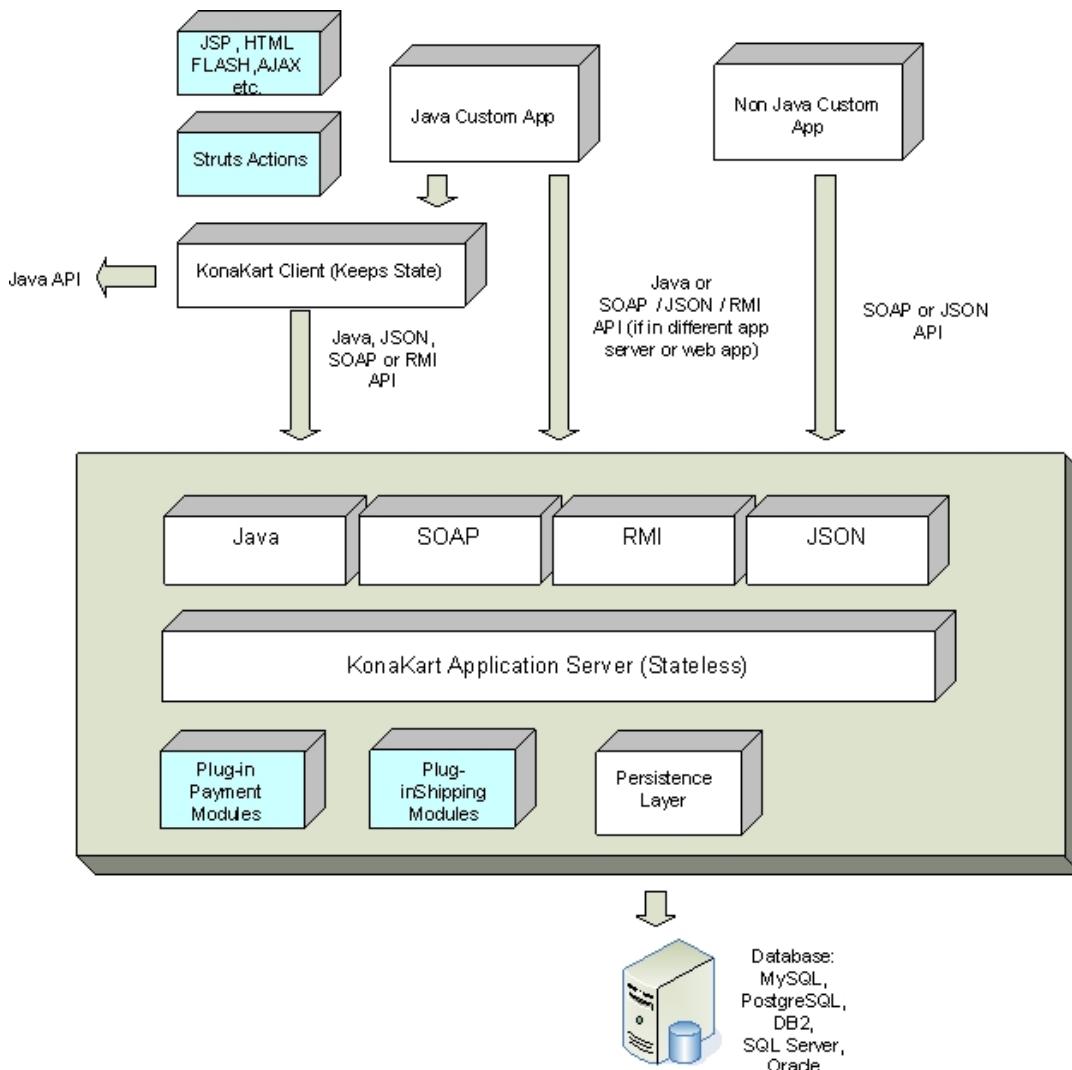
Events may be written to another database rather than the production database in order to not impact production performance. All events are written to a local queue which is emptied by a separate thread so that the insertion of events never becomes a performance bottleneck.

Chapter 4. Architecture

KonaKart has a flexible architecture lending itself to a variety of different physical implementations to suit different needs.

Software Architecture

KonaKart has a modular architecture consisting of different software layers as can be seen in the diagram below. Source code is provided for the blocks colored in light blue. The diagram shows how storefront applications written in Java and other technologies may interface to the KonaKart engines using one of the supported API technologies.



KonaKart - Software Architecture

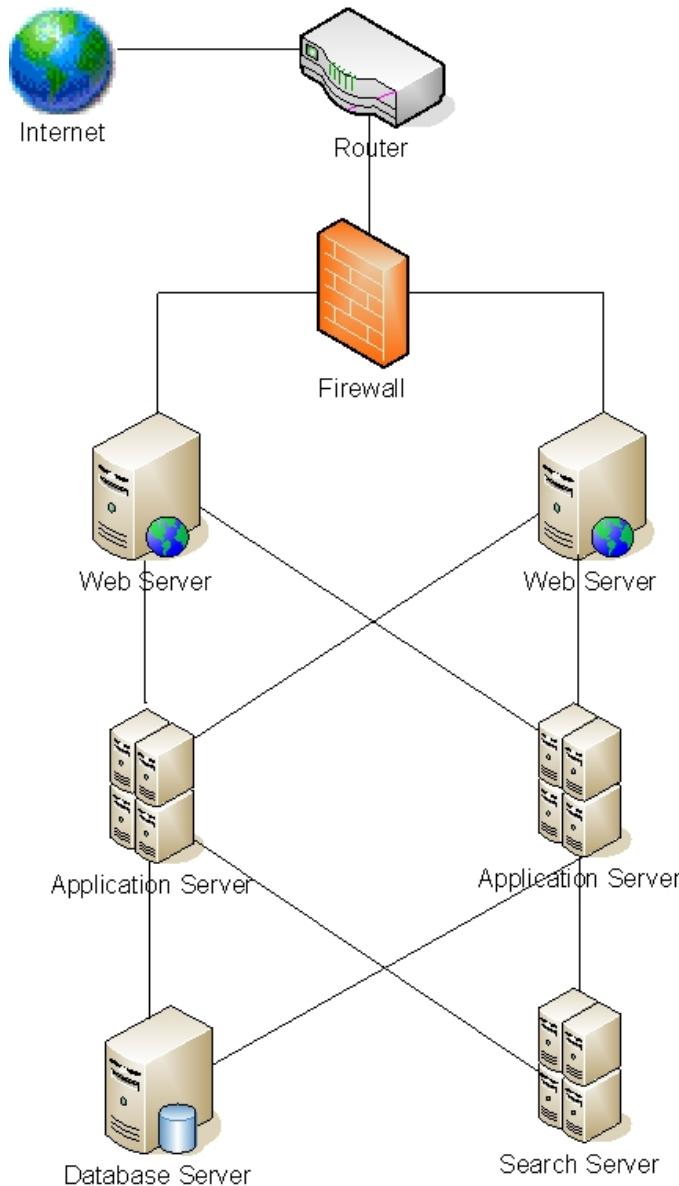
The KonaKart Server is a multi-threaded component that contains the core functionality of the eCommerce application. It exposes a SOAP Web Service interface (WSDL [http://www.konakart.com/konakart/ser-

vices/KKWebServiceEng?wsdl]), an RMI interface, a JSON interface and a Java API (Javadoc [http://www.konakart.com/javadoc/server/]). It interfaces to a persistence layer and plug in modules for calculating shipping costs, promotional discounts and for connecting to payment gateways. The persistence layer supports databases from many different vendors such as Oracle, Microsoft's SQL Server, DB2 from IBM, MySQL, PostgreSQL and many others.

The KonaKart Client manages the state of a user (associated with the user's session) as he navigates around the application. The process of writing a web based application is greatly simplified by using the API of the KonaKart client (Javadoc [http://www.konakart.com/javadoc/client/]) rather than by calling the KonaKart Server directly. Note however that you do lose some flexibility calling the Client rather than the Server since the Client has been written for a certain type of storefront application which may not exactly match the storefront that you need to build.

Struts [http://struts.apache.org/] is a popular framework that implements the Model-View-Controller (MVC) architecture. The source code of the Struts Action classes (for the store front application [http://www.konakart.com/konakart/Welcome.action]) is included in the download package in order to provide examples of how to call the KonaKart Client API. The store front application uses JSPs to generate the UI. However, different technologies can easily be implemented thanks to the modular approach of KonaKart. An example of this is the jQuery demo that uses the KonaKart jQuery library to make JavaScript calls directly to the KonaKart server engine from within an HTML page on the browser.

Deployment Architecture



KonaCart - Hardware Architecture Diagram

KonaCart software resides on the application servers. Each physical application server may contain multiple instances of KonaCart. Each instance of KonaCart can communicate with other applications through a SOAP Web Service, RMI or JSON interface. All application servers must point to the same database server which may be a cluster to provide fault tolerance.

Each web server can forward requests to any KonaCart instance. However, once a session has been established, all subsequent requests must be passed to the same instance, since it contains state information for that session. Note that sticky sessions are only necessary when using the Client Engine. The Server Engine is stateless.

Chapter 5. Installation

Please read this section carefully before attempting to install KonaKart.

Before You Begin

Before proceeding, please check that your chosen platform is one that is currently supported and that you have installed the pre-requisite software.

Platforms Supported

Currently KonaKart can be installed on Linux, Unix, or Windows XP/2003/Vista/7/8. It has been successfully installed on other platforms including Mac OS using the manual installation .

Please contact the KonaKart team at <support@konakart.com> if you would like to see KonaKart support running on another platform.

Pre-requisites

- A Java runtime environment
- A database loaded with KonaKart tables
- KonaKart itself

Install Java

KonaKart requires the Java 2 Standard Edition Runtime Environment (JRE) version 6.0 or later.

- Download the Java 2 Standard Edition Runtime Environment (JRE), release version 6.0 or later, from <http://java.sun.com/j2se> .
- Install the JRE according to the instructions included with the release.
- Ensure you have set JAVA_HOME in your environment prior to installation.

The installer attempts to locate your JRE automatically but you can override the one that's found if you require. The selected JRE is validated to help you avoid typing errors when entering the JRE location manually.

Create a Database

KonaKart needs a JDBC-compliant database. For the community edition of KonaKart you must use either MySQL (with the InnoDB engine to get support for transactions), PostgreSQL, Oracle, DB2 or MS SQL Server but if you would like KonaKart to be supported on any other database please contact us. MySQL works well with KonaKart and is free so this makes a popular choice. You can obtain MySQL from <http://www.mysql.com/>.

Check the specific notes for each database to verify that the database you plan to use is fully supported (see below in this FAQ) or whether you might have to take a few additional manual steps to load the database objects.

Once the database is installed, create a new database user for KonaKart then either run the installation wizard which will (optionally) load up the database ready for using KonaKart, or if you prefer, execute the SQL script appropriate for your chosen database manually. The database initialization scripts are provided for all supported databases under the database directory under your KonaKart installation directory.

Upgrading the Database between releases of KonaKart

Starting with the upgrade from KonaKart v2.2.0.0 to v2.2.0.1 there will always be an upgrade script provided that can be run on your database without risking the loss of your existing data (although it is always recommended to backup your database on a regular basis). The upgrade script will apply all the database changes that are required to upgrade a database being used for a specified KonaKart version to the next.

As an example a script called `upgrade_2.2.0.0_to_2.2.0.1.sql` is provided that will upgrade your database being used on a KonaKart v2.2.0.0 system to one suitable for a KonaKart v2.2.0.1 system.

If you chose to skip KonaKart releases for whatever reason, you will have to apply the upgrade scripts for all intermediate releases - upgrade scripts for all supported versions are planned to be shipped with all future releases.

Another option is to run the full database creation script (see above) which is always provided for every release. Note that this has the disadvantage of clearing away all of the data you may have set up for your KonaKart store (eg. your special configuration data, your catalogs etc) so will probably not be the preferred option for existing storekeepers.

Install KonaKart

Once you have java (java 6, java 7 or java 8) installed and a database (either pre-loaded with all the necessary tables etc or ready to be loaded), you are ready to install KonaKart itself. To do this, download an installation kit, compatible with your chosen platform, and follow the installation instructions below for your platform :

Note that if the GUI or silent installers do not work on your platform you should download the zip version of KonaKart and follow the manual installation instructions.

Installing KonaKart on Windows

Run the set-up program that executes a graphical installation wizard - see Graphical Installation Wizard below. (You can use the "Silent Mode" installation if you prefer, but the graphical version is probably easier if you're installing for the first time).

Installing KonaKart on Unix/Linux

Create a terminal session on your machine and enter the following: (You may prefer to use commands to do the same thing from your X-desktop if you have one installed).

```
$# (replace 2.2.6.0 with the version you have downloaded)
$ chmod +x KonaKart-2.2.6.0-Linux-Install
$ ./KonaKart-2.2.6.0-Linux-Install
```

If you have a graphical environment on your Linux/Unix machine you will be able to run the GUI. In which case see the Graphical Installation Wizard below (identical steps to the Windows installation).

If you don't have a graphical environment you will see this warning message:

```
$ ./KonaKart-2.1.0.0-Linux-Install
This program must be run in a graphical environment
or in silent, unattended mode (with the -S option).
```

Silent Mode Installations

When running in "silent" (-S) (or "unattended") mode you are able to specify configuration parameters on the command line, for example:

```
$ ./KonaKart-2.1.0.0-Linux-Install -S \
-DatabaseType mysql \
-DatabaseUrl jdbc:mysql://localhost:3306/mykkdb \
-DatabaseUsername kkdbusr \
-DatabasePassword ikk8271
```

Silent Mode Parameters

The following parameters can be added to the command line, as in the example above, to specify default values for KonaKart at installation time:

Parameter	Default Value	Explanation
DatabaseType	mysql	mysql, postgresql, db2net, oracle, mssql
DatabaseUrl	jdbc:mysql://localhost:3306/db-name?zeroDateTimeBehavior=convertToNull	Database URL
DatabaseUsername	root	Database User's Username
DatabasePassword		Database User's Password
DatabaseDriver	com.mysql.jdbc.Driver	Database Driver
mssqlDBO	dbo	Database Owner (only used by MS SQL Server)
InstallationDir	Windows: C:\Program Files\KonaKart *nix (as root): /usr/local *nix (as user): ~/konakart	Installation Directory
LoadDB	0	1=Load DB 0=Do not Load DB
JavaJRE		The Java runtime location
PortNumber	8780	KonaKart Port Number
HTTPSPortNumber	8783	KonaKart HTTPS Port Number

sqlEncoding	utf-8	Encoding for files
addToClasspath		Custom classpath entry

Note: the value specified for addToClasspath will be added to the installer's classpath (after a classpath separator suitable for the platform) so that it is made available when the installer executes java utilities during the installation process. This can be very handy for specifying a JDBC driver that isn't shipped with KonaKart. This could be useful, for example, for specifying a Microsoft SQL Server JDBC driver for setting up the KonaKart data in a MS SQL Server database.

Note: Only set sqlEncoding if you have encoding problems during the installation on your particular platform. It is not normally required to be set. If you need to set it, choose an encoding that your platform can use (many choices are available including: cp860 cp861 cp862 cp863 cp864 tis-620 cp865 cp866 cp869 dingbats macCentEuro cp874 macUkraine jis0201 macThai iso8859-10 iso2022-jp iso2022 macIceland iso8859-13 iso8859-14 cp737 iso8859-15 iso8859-16 Symbol macRomania gb1988 iso2022-kr macTurkish macGreek ascii cp437 macRoman iso8859-1 iso8859-2 iso8859-3 macCroatian iso8859-4 koi8-r ebcdic iso8859-5 cp1250 macCyrillic iso8859-6 cp1251 iso8859-7 cp1252 koi8-u macDingbats iso8859-8 cp1253 iso8859-9 cp1254 cp850 cp1255 cp1256 identity cp852 cp1257 cp1258 utf-8 cp855 cp775 symbol cp857 unicode).

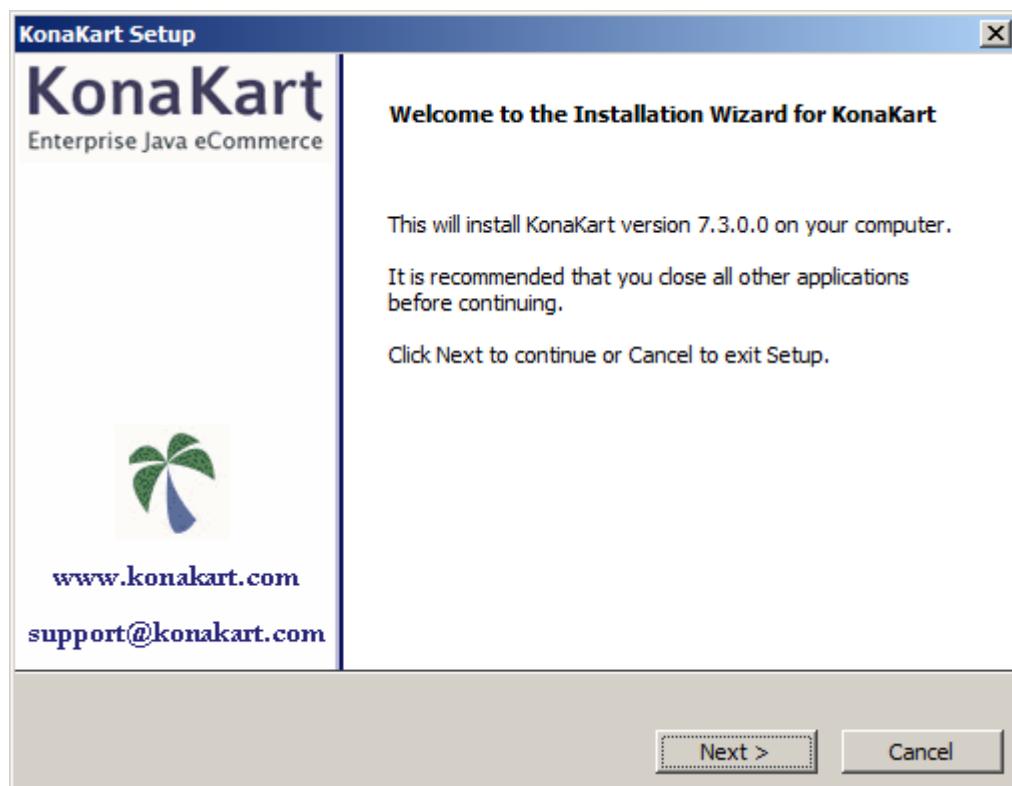
Graphical Installation Wizard

This shows a typical installation that uses the wizard:

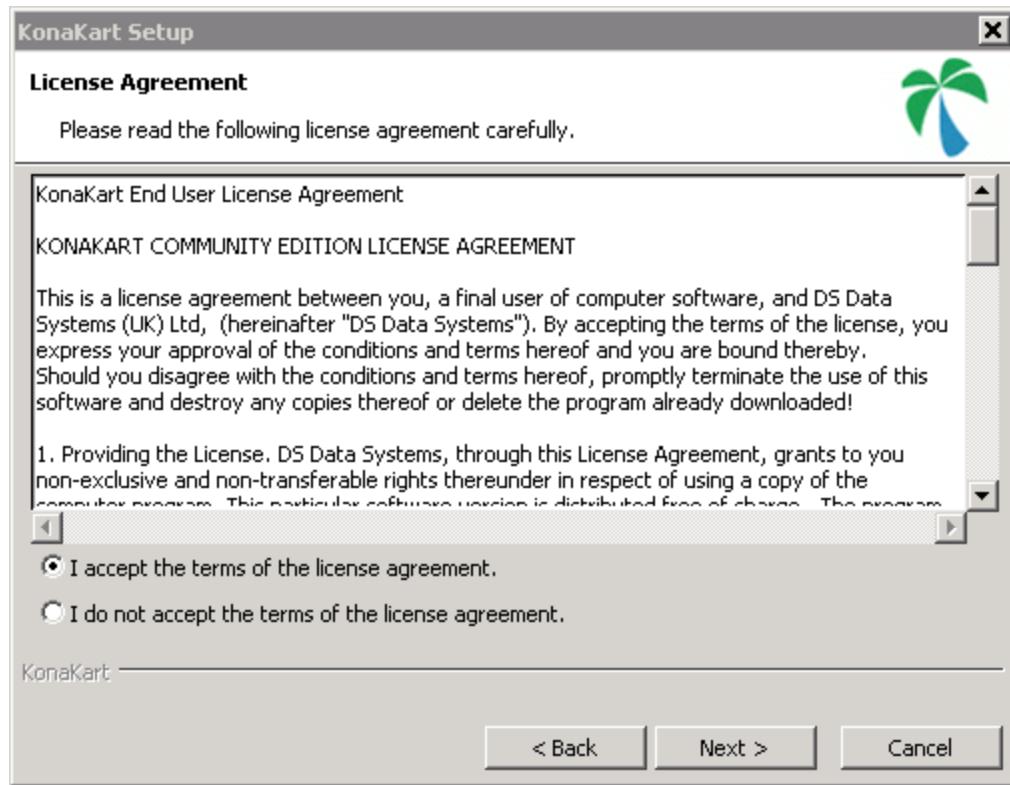
Either double-click on the installation setup program (either KonaKart-2.2.0.4-Windows-Setup.exe on Windows, or KonaKart-2.2.0.4-Linux-Install on Linux - or respective later version numbers) or run it from a command shell (with optional arguments as documented above for the Silent Mode Install). You are first presented with this small window which allows you to confirm that you wish to proceed with the installation:



Click on Yes to continue to:

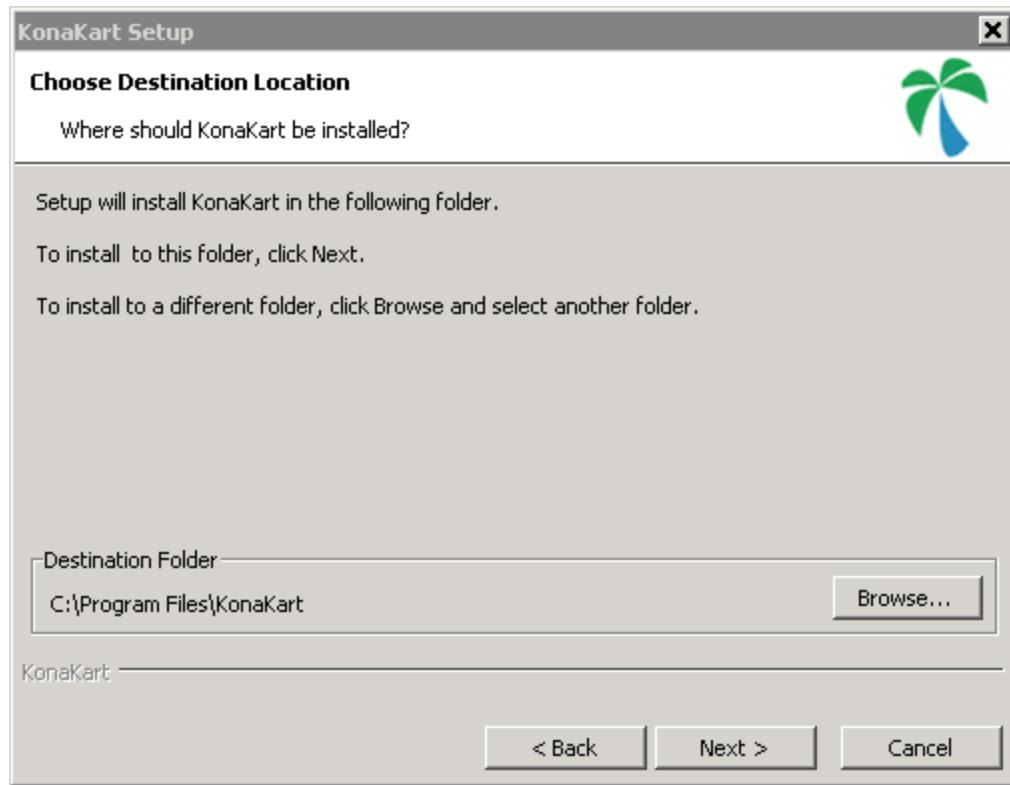


Check that you have the correct version number and click on next to get to the next screen. Note in passing the email address for support questions. Please contact us and / or search the forum if you have any difficulties at all with the installation and we will endeavour to help you as soon as possible.



Please read the license agreement carefully and if you are happy to do so under the terms of the agreement, click on the "I accept the terms of the license agreement" bullet and click next to continue. If you are not prepared to accept that license agreement please quit the installation at this point.

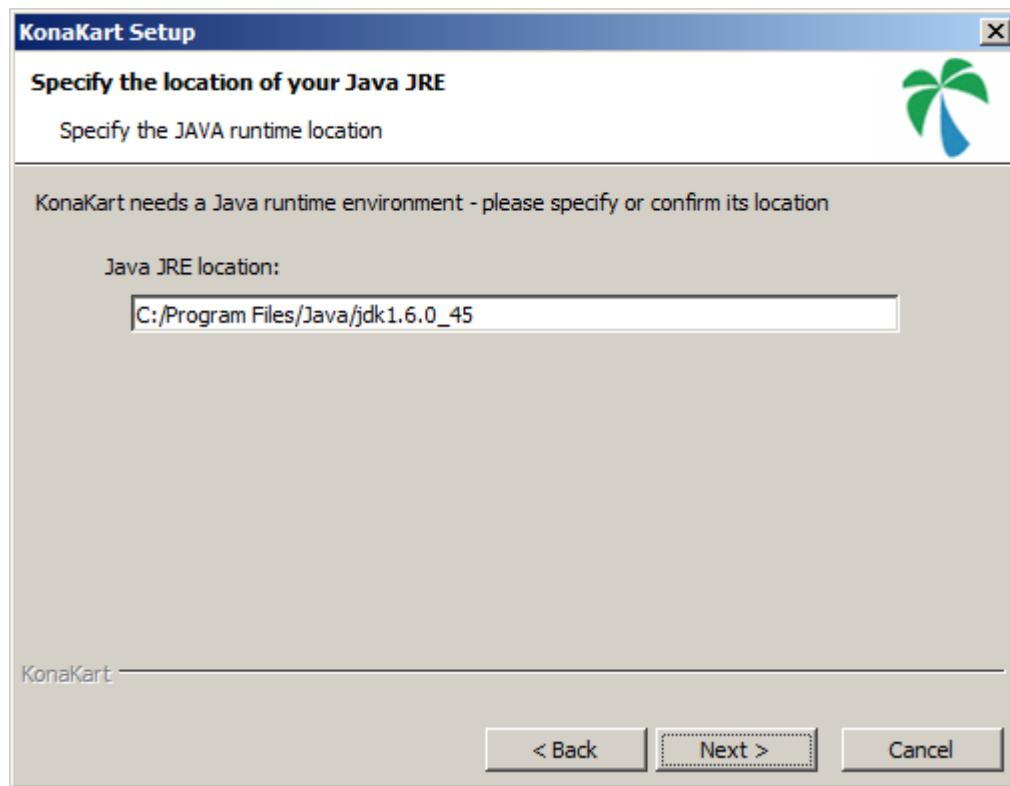
Click on next to reach:



This is where you specify where you want KonaKart to be installed. On Windows this defaults to "C:\Program Files\", on Linux this is the user's home directory (if the user is not root) or "/usr/local" (if the user is root). This can be specified in the silent mode of on the command line of the GUI version using -DInstallationDir.

It is recommended that you accept the default location, but this is not essential.

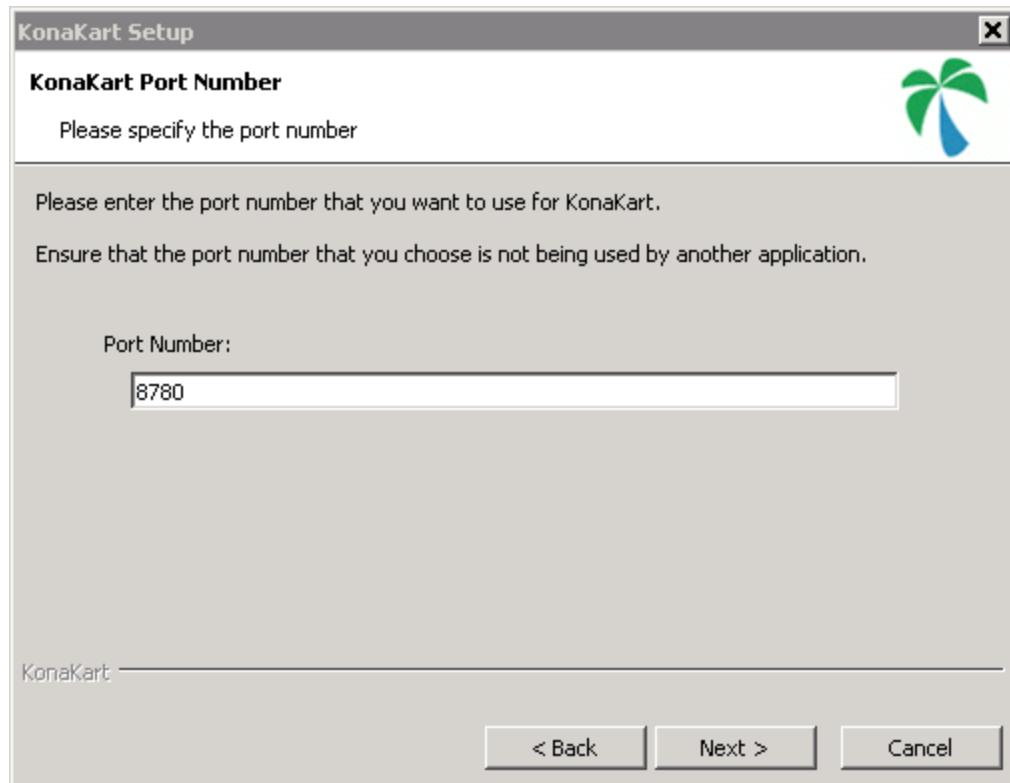
On clicking next you reach:



Here you have to confirm or specify the location of the java runtime environment. The wizard will try to find this for you but it is not always successful. In the cases where it isn't successful (or you wish to change its selection) you will have to enter the location manually. If you have installed java v6 in the default location or it appears in your environment's path, the wizard should find it for you.

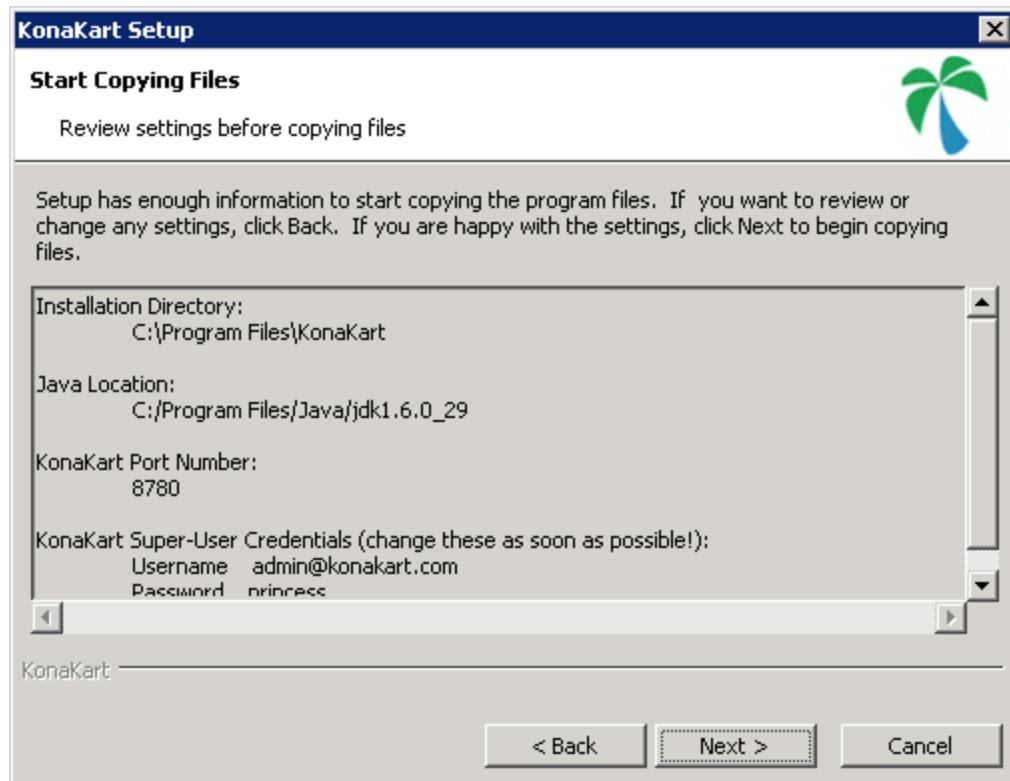
The java location selected is validated to help you avoid typing errors and will only allow you to proceed in the wizard when it validates the location successfully.

Click on next to get to:



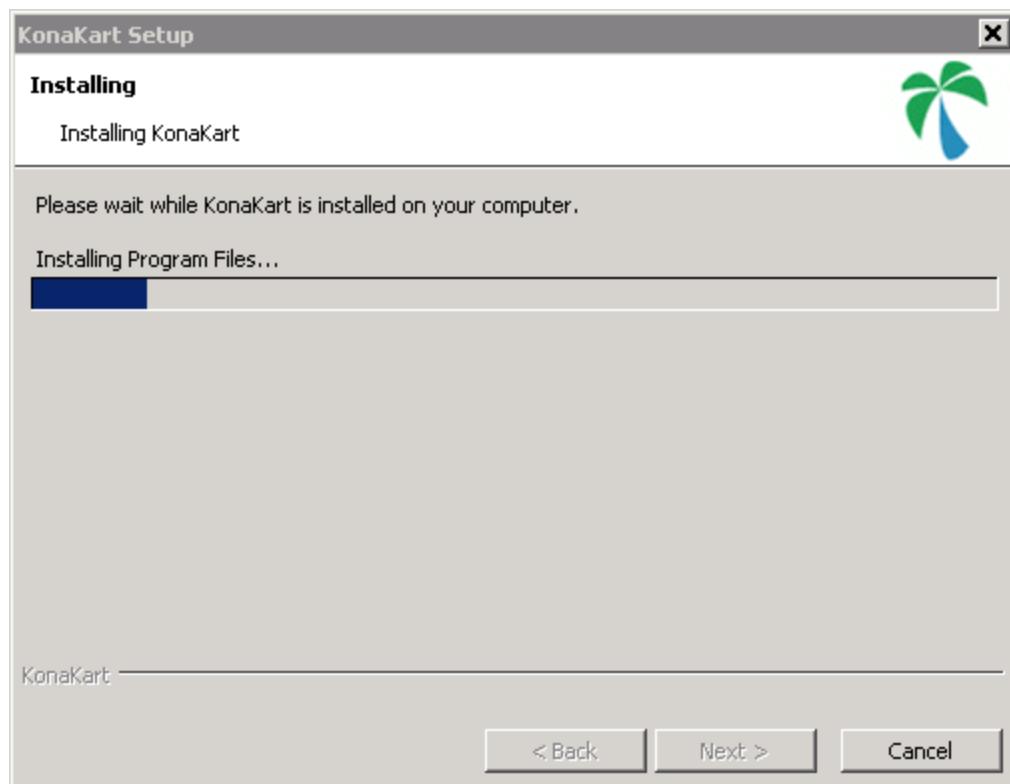
This is where you define the port number that KonaKart will run on. Actually, KonaKart uses Apache Tomcat, so this is the port number that Tomcat is configured to run on. Whilst it is certainly possible to choose another port, it's advisable to accept the default which is 8780. KonaKart will not start up if another application is using the port you select, so make sure the port you select here is not currently being used.

Clicking next moves you on to:

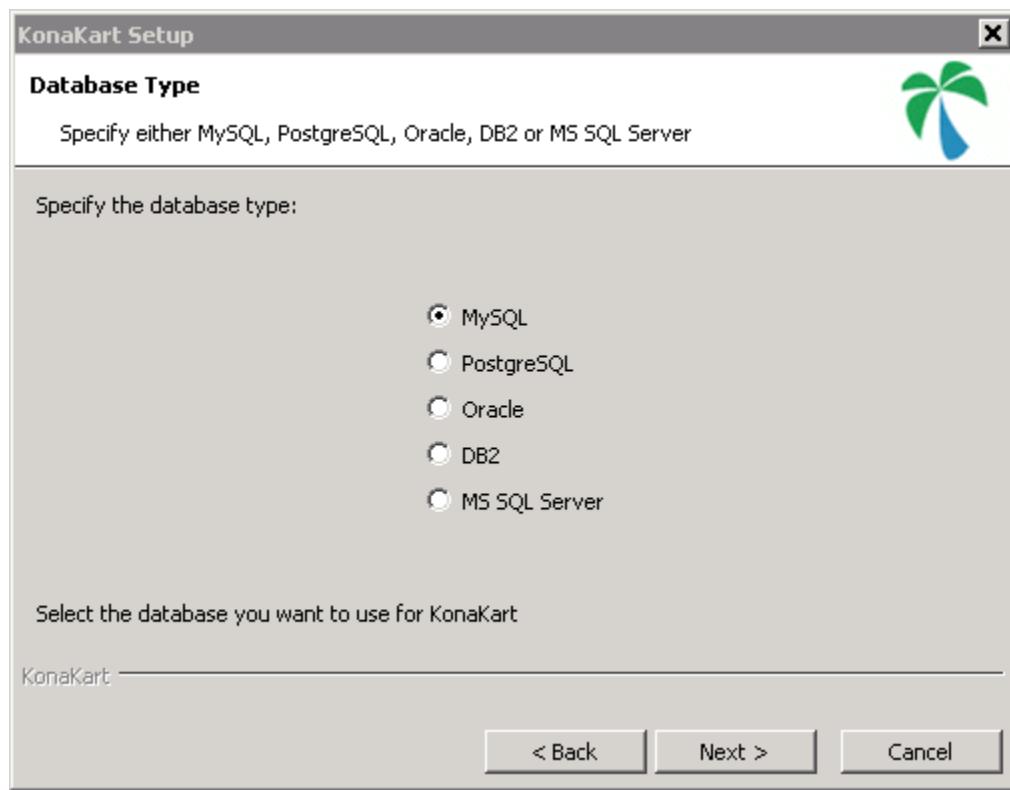


This is your final chance to check your settings before copying the files into position.

Clicking next will start the process of copying the KonaKart files into position as can be seen on the next screen:

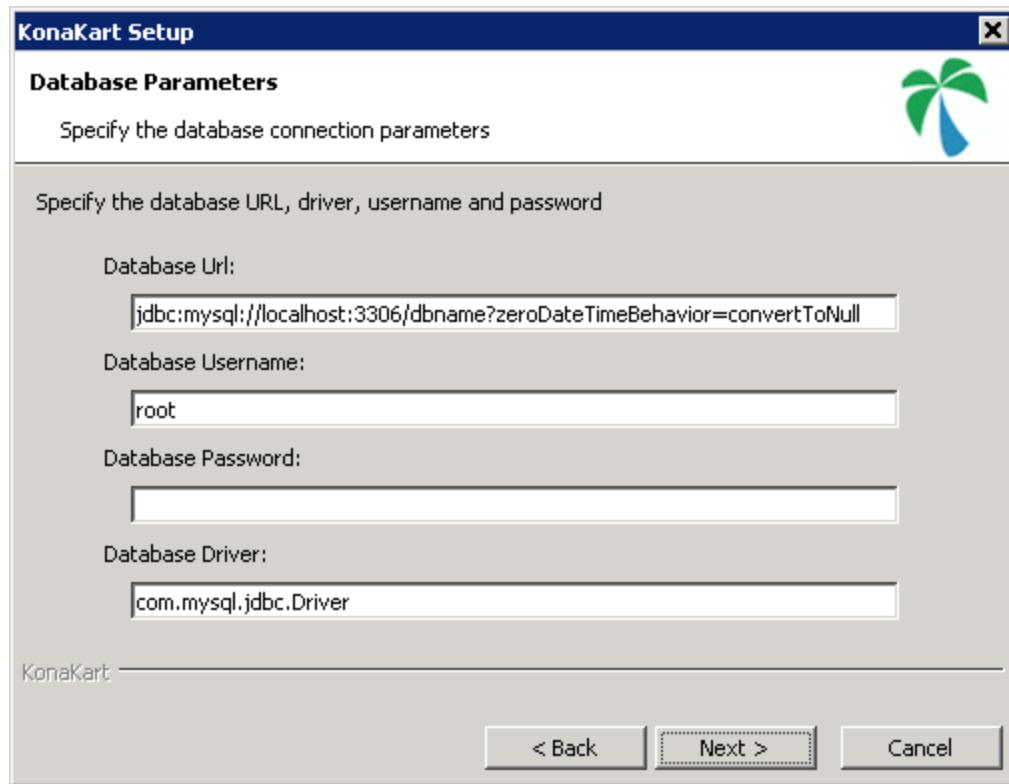


After about a minute or so the file copying will have completed and you are presented with this screen to choose the Database Type that you wish to use:



Choose your preferred database type. You must set up the database yourself - this is not done as part of the installation (see database creation).

Click next to continue to :

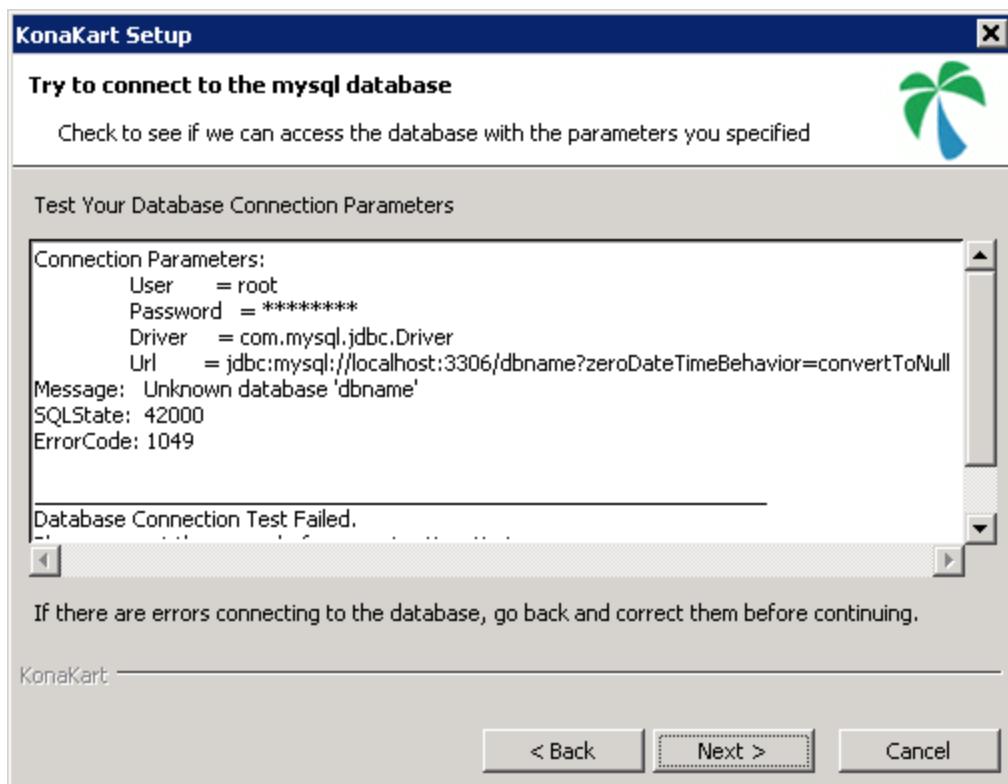


It is very likely that you will have to modify values on this screen. You have to define the database connection parameters to KonaKart for the database that you created earlier (see database creation)

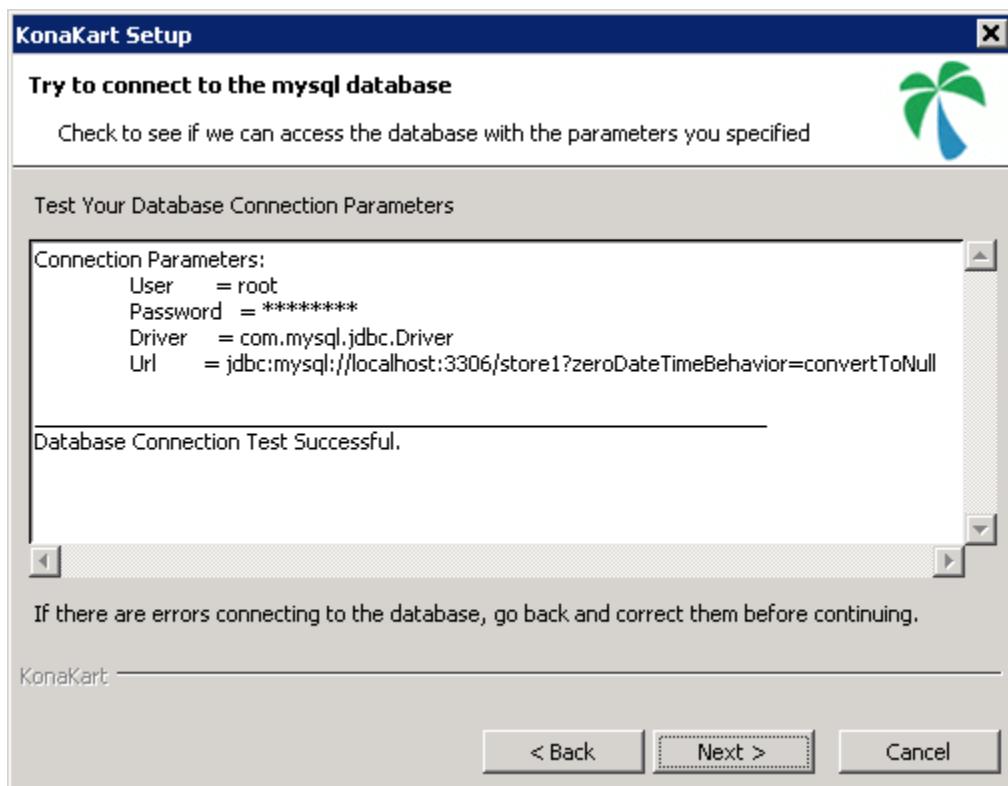
KonaKart currently supports MySQL, PostgreSQL, Oracle, DB2 and MS SQL Server and includes all the JDBC driver jars required to access these.

Note that you must append "?zeroDateTimeBehavior=convertToNull" to your Database URL if you're using MySQL. Typically, for MySQL, you will need to change "dbname" in the default URL for the name of your own database schema. A good example for such a name might be "store1" or "konakart" but you are free to choose whatever name you like.

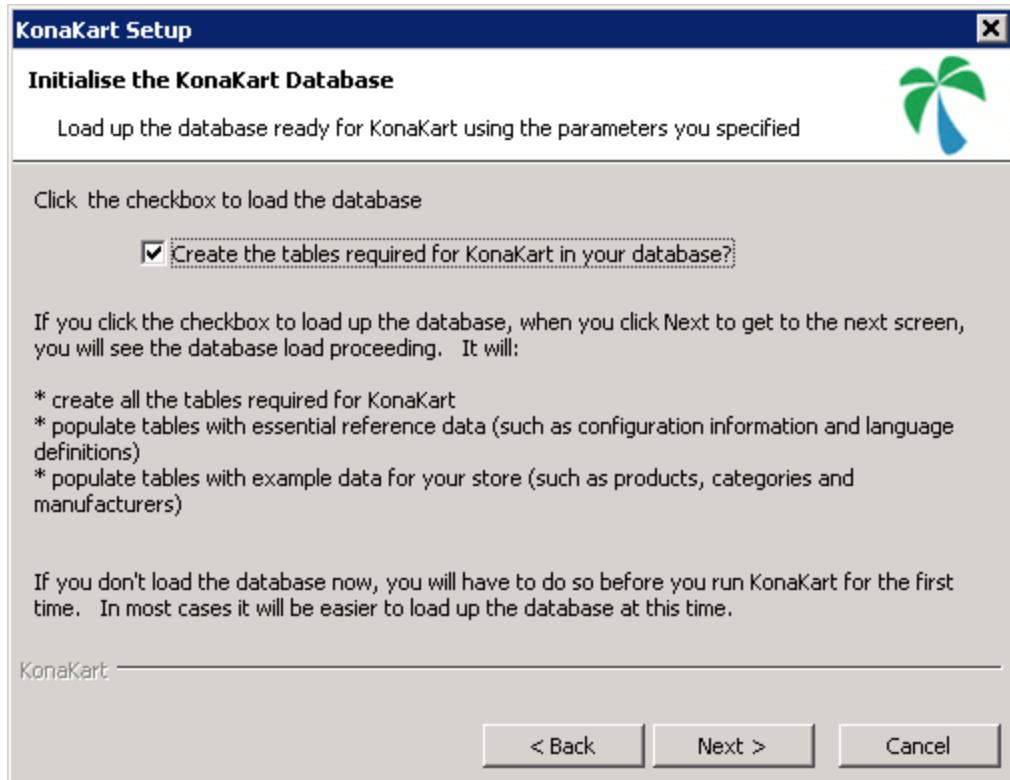
After clicking next, the installer will check the database connection and report the results on the next screen as follows. For an unsuccessful connection you will see something like this:



At this point you can go back and modify the database connection parameters and then click next to try the database connection test again. If successful you will see a screen like this:



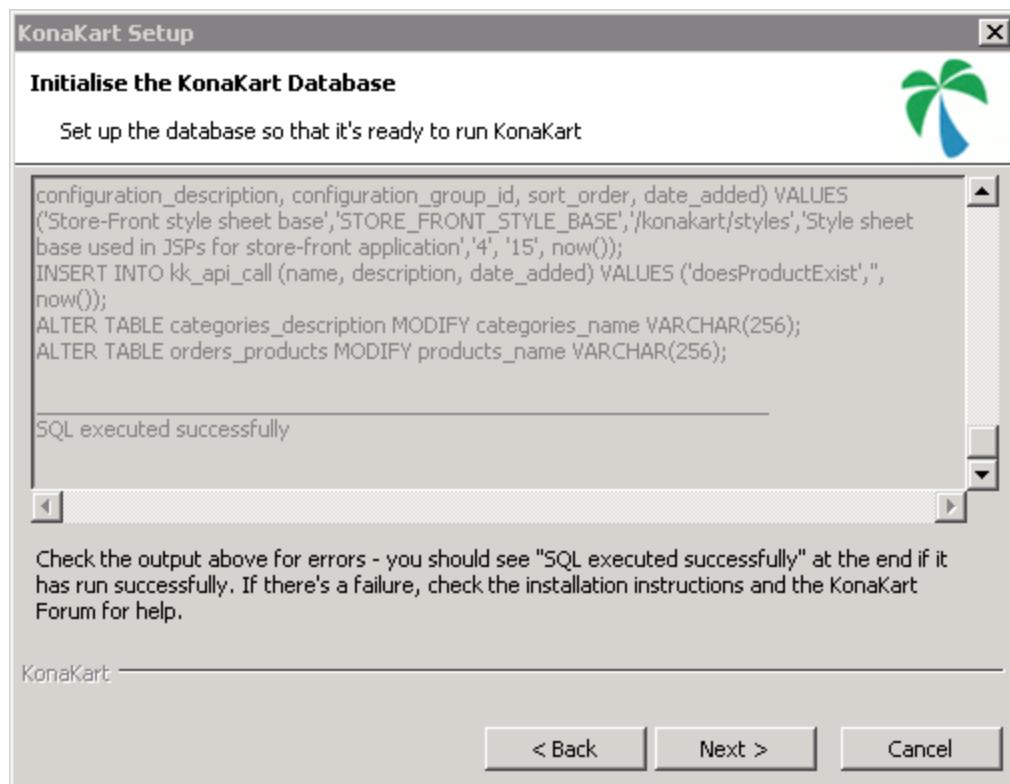
If the database connection fails, you can choose to click on next and finish the installation without executing the database initialisation. Normally, for a fresh install, you are advised to correct your database connection parameters so that you see a successful database connection message, then proceed to the database initialisation screen which looks like this:



Note that the default is that the database initialisation is NOT executed. To execute it you must click on the checkbox and click next to reach the next step.

Be warned that the database initialisation script will drop and re-create all the KonaKart tables and populate them with a default set of starting data. This default starting position is ideal for users who are setting up KonaKart for the first time but this is probably not what you want to do if you already have a KonaKart database with a product catalog loaded for your store.

If you do not check the 'Create the tables..' box then click next you would jump to the final page of the wizard. Alternatively, if you do check the 'Create the tables...' box then click next, you will be presented with a screen that shows the loading of the database initialisation script in a scolling window which will look like this:



In the above example the script ran successfully (see last comment "SQL executed successfully" in the scrollable text area). If the script does not run successfully you will see the error message in this window.

After clicking next you reach the following screen:

Finally you are congratulated on a successful installation on the final screen of the wizard:



Finally you have the option to create a desktop icon (which is defined to start the KonaKart server and launch the GUI) and launch the application immediately after the installation has completed. The "Launch KonaKart" option executes a startup of the KonaKart server and then launches the default browser to show the KonaKart UI, and the KonaKart Administration Application.

Manual Installation

If you are installing on a platform that supports the GUI installer (Windows, Linux, Unix), it is recommended that you use that. If not, but you are on a platform that supports the silent form of the installer (again Windows, Linux, Unix), it is recommended that you use that. Otherwise, use the manual installation.

If you plan to install KonaKart in an existing servlet container, it is still advisable to run through the GUI installer if you can. The reason for this is that it will populate all the properties files for you and load your database automatically. Once you have done this you can make WARs from the GUI-installed version of KonaKart (details below) and deploy them elsewhere as you please.

The KonaKart Installation section contains general KonaKart installation instructions (although focuses on using the GUI-driven and silent versions of the installer) and contains important information that is also relevant for the manual installation so check this before starting out.

In general, you need to follow all the documented installation instructions except for the "Install KonaKart" section which explains how to use the automated GUI and Silent versions of the installation.

Perform all the documented installation instructions for:

- A Java runtime environment
- A database loaded with KonaKart tables

For the purposes of this FAQ we'll use MySQL and a database named "konakart".

Using a downloaded zip file for manual installation from the Downloads [<http://www.konakart.com/downloads>] page on the KonaKart website unzip to a suitable location and then follow the following steps:

1. Populate the Database

Make sure you have created an empty database instance in the RDBMS of choice as per the FAQ instructions: eg:

```
$ mysql> create database konakart
```

To create the database load konakart_demo.sql for the database you have chosen. In this case:

```
$ mysql -u root -p konakart < ./konakart/database/MySQL/konakart_demo.sql
```

2. Copy the Duplicated Jars

For users of the "zip" installations - take note!

In order to minimize the size of the download files quite a few of the jars are not copied into multiple webapps where they are required to run KonaKart successfully. A platform-specific script is provided that will copy the duplicated jars (and a few other file types) from the konakart webapp to the other webapps. The appropriate ("bat" or "sh") script must be run to complete a successful installation. The scripts (both called "copyDuplicates" can be found in the custom directory under the installation home. Both scripts require a single parameter which is the installation home directory. For example, on Windows:

```
C:\Program Files\KonaKart\custom>copyDuplicates C:\Program Files\KonaKart
C:\Program Files\KonaKart\custom>REM
C:\Program Files\KonaKart\custom>REM Copy duplicate files
C:\Program Files\KonaKart\custom>REM
```

3. Set Database Parameters

Set DB name (and other database connection parameters) in *konakart.properties* and *konakartadmin.properties*. Change the string "dbname" to the name of your database (in this case "konakart") in the following files:

{konakart}/webapps/konakart/WEB-INF/classes/konakart.properties

{konakart}/webapps/konakartadmin/WEB-INF/classes/konakartadmin.properties

This is documented in the "Defining the Database Parameters" section below.

4. Deployment

Now, when it comes to deployment, there is a choice.

Either: run KonaKart using the tomcat that was provided in the download kit or create WARs and load them into your chosen servlet container.

a. Deploy to the tomcat provided

- i. Start tomcat using {konakart}/bin/startkonakart.sh (or, if you are working on Windows, use {konakart}\bin\StartAndLaunchKonaKart.bat).

- ii. Run KonaKart and KonaKartAdmin

Try <http://localhost:8780/konakart/> [<http://localhost:8780/konakart/>] and <http://localhost:8780/konakartadmin/> [<http://localhost:8780/konakartadmin/>] in your browser.

- b. Create WARs and deploy in another servlet container

- i. Generate and deploy the war files.

```
$ cd {konakart}/custom
$ ./custom/bin/ant make_wars
```

More detailed instruction for building the war files are available in the customization section if you need them.

Deploy the generated WAR files. This is different for different servlet containers, but for tomcat, just copy the .war files in {konakart}/custom/war to your tomcat's *webapps* directory.

Restart the servlet container if necessary.

- ii. Run KonaKart and KonaKartAdmin

Try <http://localhost:8780/konakart/> [<http://localhost:8780/konakart/>] and <http://localhost:8780/konakartadmin/> [<http://localhost:8780/konakartadmin/>] in your browser, adjusting the port as necessary if you deployed to a different port number.

Starting Up and Shutting Down KonaKart

If you are not using the default bundled tomcat refer to your chosen servlet container's startup/shutdown procedures. If you have installed the default KonaKart system with a bundled tomcat, you can follow these instructions to startup and shutdown KonaKart:

Starting up KonaKart

The KonaKart Server can be started by executing the following commands:

<pre>c:\> %CATALINA_HOME%\bin\startkonakart.bat</pre>	<small>(Windows)</small>
<pre>\$ \${CATALINA_HOME}/bin/startkonakart.sh</pre>	<small>(Unix/Linux)</small>

On Windows there are also shortcuts created under the KonaKart program group that can be used to:

- start the KonaKart Server
- stop the KonaKart Server
- launch the KonaKart application in your default browser
- launch the KonaKart Administration application in your default browser

- uninstall KonaKart

Shutting down KonaKart

KonaKart can be shut down by executing the following command:

```
c:\> %CATALINA_HOME%\bin\stopkonakart.bat          (Windows)
$ ${CATALINA_HOME}/bin/stopkonakart.sh           (Unix/Linux)
```

On Windows a shortcut is created under the KonaKart program group that can be used to shut down the KonaKart server.

Setting up KonaKart as a Windows Service

You can run KonaKart as a Windows Service if you wish. A special service.bat file is provided (in the KonaKart/bin directory) to make the installation and removal of KonaKart as a service extremely straightforward.

To install KonaKart as a service run:

```
REM -----
REM To install KonaKart as a Windows Service
REM -----
C:\Program Files\KonaKart\bin>service install
Installing the service 'KonaKart' ...
Using CATALINA_HOME:      C:\Program Files\KonaKart
Using CATALINA_BASE:       C:\Program Files\KonaKart
Using JAVA_HOME:           C:\Java\jdk1.6.0_12\
Using JVM:                C:\Java\jdk1.6.0_12\jre\bin\server\jvm.dll
The service 'KonaKart' has been installed.
```

To remove KonaKart as a service run:

```
REM -----
REM To remove KonaKart as a Windows Service
REM -----
C:\Program Files\KonaKart\bin>service remove KonaKart
The service 'KonaKart' has been removed
```

Note that the tomcat7.exe provided in the bin directory is a 64' executable. This needs to work with a 64' version of Java (a 64' version of the jvm.dll). If you do not have a 64' version of java you can replace the tomcat7.exe with a 32' version and use the 32' version of java. You can use the supplied tomcat7w.exe client tool to modify the system properties for your service.

To configure KonaKart as a service run:

```
REM -----
REM To configure KonaKart as a Windows Service
REM -----
C:\Program Files (x86)\KonaKart\bin>tomcat7w //ES/KonaKart
```

Note that you may need to configure a compatible jvm.dll and suitable memory settings for your environment. If the service fails to start or runs out of memory these are signs that you need to review the service configuration.

Default Admin App Credentials

Three users are created with different roles assigned.

Username	Password	Roles
admin@konakart.com	princess	KonaKart Super-User
doe@konakart.com	password	Sample User
cat@konakart.com	princess	KonaKart Catalog Maintainer
order@konakart.com	princess	KonaKart Order and Customer Manager

Choose new usernames and passwords to secure the KonaKart Administration Application at the earliest opportunity.

The default users and roles are set up as examples of typical configurations of the role-based security system. You may wish to add new Admin users or adjust some of the roles as you see fit. The three users above should give you a few ideas about how the system can be configured.

The three users above are defined as "Admin Users". Note that "Admin Users" can actually log in to the KonaKart store using the same credentials. It doesn't work the other way around however: "Non Admin Users" cannot log into the Admin Application.

Although it's not recommended, it is possible to disable security completely if you wish. To configure this, see the comments inside the konakartadmin.properties file which can be found under the konakart installation directory at:

webapps\konakartadmin\WEB-INF\classes\konakartadmin.properties

(There are a number of additional configuration options that you can adjust to modify the behaviour of the KonaKart Administration Application with respect to security - please refer to the comments in the above properties file for details).

Admin Password Validation

Admin passwords can be validated against a set of configurable rules defined in the konakartadmin.properties file. The options for password validation are as follows:

```

# Min/Max length for Admin Passwords
#(set min password length for Admin app to match - default also 8)

konakart.password.minimumChars          = 8
konakart.password.maximumChars          = 20

# An upper case character is any character in A..Z
konakart.password.mustContainUpperCase = true

# A lower case character is any character in a..z
konakart.password.mustContainLowerCase = true

# A numeric character is any character in 0..9
konakart.password.mustContainNumeric   = true

# A "special" character is any character that is not a..z, A..Z, or 0..9
konakart.password.mustContainSpecialChar = true

# When a password is changed the new one can't be the same as any of the previous N
# Set to -1 to not carry out this check
konakart.password.mustDifferFromLastNPasswords = 4

# Login will not be successful if the password has expired
# Admin App users will be forced to change their password if it has expired
# Set to -1 if you don't ever want passwords to expire
konakart.password.expiryDays           = 90

```

Note that from the 7.3.0.0 release of KonaKart the Admin passwords will, by default, expire after 90 days. If you do not want this behaviour, modify the konakart.password.expiryDays property as required.

If an Admin App user attempts to log in with a password that has expired a dialog box will appear which allows the user to change that password.

An Admin App user can use the Change Password option when inside the Admin App to change his password at any time.

Remember that the admin usernames may be used in other areas in the KonaKart system (eg. for quartz jobs) so remember that these passwords can also expire according to the same rules as for Admin App users.

Super User

A "Super User" must be an Admin User with a role that has the super_user indicator set.

Since the "Super User" has privileges to change all the configuration settings in a KonaKart store, you must guard these credentials carefully.

Ensure that you change the password of the "Super User" account(s) as required by your Site Security policy for highly-privileged accounts.

Installation Notes for Databases

Defining the Database Parameters

You define the database parameters in two configuration files underneath your konakart installation directory at:

```

webapps\konakart\WEB-INF\classes\konakart.properties
and
webapps\konakartadmin\WEB-INF\classes\konakartadmin.properties

```

(Therefore, the default on Windows will be at C:\Program Files\KonaKart\webapps\konakart\WEB-INF\classes\konakart.properties).

Inside these files you will find: (the url parameter is broken into two lines for readability only - you should keep it on one line)

```
# -----
# D A T A B A S E   P R O P E R T I E S
# Database Connection Parameters Set by Installer on 20-Jan-2009
# -----

torque.applicationRoot = .

torque.database.default          = store1

torque.database.store1.adapter    = mysql
torque.dsfactory.store1.connection.driver = com.mysql.jdbc.Driver
torque.dsfactory.store1.connection.url =
        jdbc:mysql://localhost:3306/store1?zeroDateTimeBehavior=convertToNull
torque.dsfactory.store1.connection.user = fruit
torque.dsfactory.store1.connection.password = secret
```

Leave the torque.database.default equal to store1.

You need to set the five parameters appropriate for your environment:

- torque.database.store1.adapter (either "mysql", "oracle", "db2net", "mssql" "postgresql")
- torque.dsfactory.store1.connection.driver (All JDBC drivers for the supported databases are on the default classpath)
- torque.dsfactory.store1.connection.url (keep the value on the same line after the equals sign)
- torque.dsfactory.store1.connection.user
- torque.dsfactory.store1.connection.password

Encrypting the Database Parameters

If you wish you can encrypt the database credentials in the KonaKart properties files by using the CreatePassword utility that is provided in the Enterprise Extensions.

You can obtain usage information for the utility by issuing the "-?" parameter as follows:

```
@blackburn:~/konakart/utils/createPassword: ./createPassword.sh -?
=====
KonaKart Create Password Utility
=====

Usage: CreatePassword
      -f properties-file      - the properties file where the results
                                will be written
      -k                      - the key of the property to encrypt
      -p new-value            - the new-value to encrypt (typically,
                                this will be the username or password)
      [-en file encoding]     - file encoding (default is ISO-8859-1)
      [-ce console encoding]  - console encoding (default is utf-8)
      [-d]                    - outputs debug information
      [-?]                   - shows this usage information
```

A ReadMe.txt file is provided in the createPassword directory with instructions on how to use the utility.

After running the utility you end up with an encrypted password in the specified properties along with an associated encryption key.

KonaKart itself has been modified to recognise and decrypt these encrypted passwords by using the associated encryption keys.

If you create the encrypted passwords for copying into another properties file elsewhere, always remember that the encrypted database password needs its associated encryption key - so remember to copy both properties! These associated encryption key values are written on the line following the property itself so they are easy to locate.

If you specify the "-p new-value" parameter the utility will encrypt the specified "new-value". If you do not specify the "-p new-value" parameter at all, the utility will encrypt the current value of the property.

Note that it is also possible to encrypt the database username in exactly the same way as the password. KonaKart will also handle encrypted usernames in the properties files.

Defining the Database Parameters - Using JNDI

It is also possible to define your data source using JNDI. (For various optional configurations not covered here please refer to the Torque Database Configuration documentation on the Apache site.)

In this example Torque is configured to create a JNDI Data Source and deploy it into JNDI:

You would modify your konakart.properties files by commenting out the connection properties used by the default SharedPoolDataSourceFactory factory and replace them as follows:

```

torque.dsfactory.store1.factory      = org.apache.torque.dsfactory.JndiDataSourceFactory
torque.dsfactory.store1.jndi.path    = jdbc/konakart
torque.dsfactory.store1.jndi.ttl     = 60000
torque.dsfactory.store1.datasource.classname = org.apache.commons.dbcp.BasicDataSource

torque.database.store1.adapter       = mysql

torque.dsfactory.store1.jndi.java.naming.factory.initial = \
                                                org.apache.naming.java.javaURLContextFactory

torque.dsfactory.store1.datasource.driverClassName = com.mysql.jdbc.Driver
torque.dsfactory.store1.datasource.url      = \
                                                jdbc:mysql://localhost:3306/store1?zeroDateTimeBehavior=convertToNull
torque.dsfactory.store1.datasource.username = fruit
torque.dsfactory.store1.datasource.password = secret

#For JNDI you also need to comment out the following line (as it has been replaced above):
#torque.dsfactory.store1.factory=org.apache.torque.dsfactory.SharedPoolDataSourceFactory

```

In this example Tomcat (v7.0.54) is configured to create a JNDI Data Source and deploy it into JNDI and the konakart.properties definition binds to that existing data source: (Note that different definitions will be required for other application servers so refer to the documentation on those for detailed configuration specifications).

In conf/context.xml:

```

<Resource name="jdbc/villa1" auth="Container" type="javax.sql.DataSource"
maxActive="100" maxIdle="30" maxWait="10000"
username="kkUser" password="fred" driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost:3306/villa1?zeroDateTimeBehavior=convertToNull"/>

```

In conf/context.xml:

```

torque.dsfactory.store1.factory          = org.apache.torque.dsfactory.JndiDataSourceFactory
torque.database.default                 = store1

torque.dsfactory.store1.jndi.path       = java:/comp/env/jdbc/villal
torque.dsfactory.store1.jndi.java.naming.factory.initial \
                                         = org.apache.naming.java.javaURLContextFactory
torque.dsfactory.store1.jndi.java.naming.factory.url.pkgs \
                                         = org.apache.naming
torque.dsfactory.store1.jndi.ttl        = 60000

#For JNDI you also need to comment out the following line (as it has been replaced above):
#torque.dsfactory.store1.factory=org.apache.torque.dsfactory.SharedPoolDataSourceFactory

```

For the database definition for the BIRT reporting engine you need to leave the original connection properties either in a separate properties file or in konakartadmin.properties as follows:

```

# Leave these for the BIRT reports

torque.dsfactory.store1.connection.driver  = com.mysql.jdbc.Driver
torque.dsfactory.store1.connection.url     = \
                                         jdbc:mysql://localhost:3306/store1?zeroDateTimeBehavior=convertToNull
torque.dsfactory.store1.connection.user    = fruit
torque.dsfactory.store1.connection.password = secret

```

Note that if you chose to use a separate file, remember to add that file name to the "var dbPropsFile" definition in the "...\\webapps\\birtviewer\\reports\\lib\\konakart.rptlibrary" file.

Notes for DB2 and Oracle

In addition to the settings above, you have to set the validationQuery for DB2 and Oracle slightly differently to the others, as follows. This section is defined just below the database parameter definitions in the two properties files (konakart.properties and konakartadmin.properties):

```

# The SQL query that will be used to validate connections from this pool before
# returning them to the caller. If specified, this query MUST be an SQL SELECT
# statement that returns at least one row.
# Recommended settings:
# for MySQL/PostgreSQL/MS SQL use: SELECT 1
# for Oracle                  use: SELECT 1 from dual
# for DB2                     use: SELECT 1 FROM sysibm.sysdummy1

torque.dsfactory.store1.pool.validationQuery=SELECT 1
#torque.dsfactory.store2.pool.validationQuery=SELECT 1
-----
```

Notes for Postgresql

Note that the SQL that is optionally run at installation time uses the "DROP TABLE IF EXISTS TABLE-NAME;" command. This works fine for PostgreSQL 8.2 and above (which support "IF EXISTS") but not for earlier versions.

This is only a problem during the installation process; KonaCart performs well on versions of PostgreSQL prior to and after 8.2. To workaround this problem, for example for PostgreSQL 8.1, you will have to edit the database/konakart_demo.sql file and remove all the "DROP TABLE" commands then run this SQL

manually. In addition to modifying the "IF EXISTS" syntax you will also have to add SQL statements to create sequences for all the SERIAL primary keys. For example, for the counter table, which is created like this:

```
CREATE TABLE counter (
    counter_id SERIAL,
    startdate char(8),
    counter integer,
    PRIMARY KEY (counter_id)
);
```

You have to create the SEQUENCES with these conventions:

```
CREATE SEQUENCE <table>_<SERIAL column>_seq
```

for example:

.. for the counter_id column in the counter table above, you need to create a SEQUENCE like this:

```
CREATE SEQUENCE counter_counter_id_seq
INCREMENT 1
MINVALUE 1
MAXVALUE 9223372036854775807
START 1
CACHE 1;
```

Another, preferable, alternative is to update to the latest version of PostgreSQL and run the standard GUI installation and have all the data loaded by the installer.

Notes for MySQL

Note that the SQL that is optionally run at installation time uses the "DROP TABLE IF EXISTS" syntax. This works fine for MySQL 5 and above, but not on MySQL 4.1.

This is only a problem during the installation process; KonaKart performs well on MySQL 4.1 (and above) but you have to modify the konakat_demo.sql file and run it yourself manually. You have to edit the database/konakart_demo.sql file and remove all the "DROP TABLE" commands then run this SQL manually. After successfully running this SQL you will be able to run KonaKart with MySQL 4.1.

Notes for Microsoft SQL Server

The default database definition is fine if you don't need to support double-byte unicode characters in every column. If you do need to support double-byte unicode (eg. Chinese characters) in additional columns you will need to convert the selected varchar columns to nvarchar columns. This isn't done by default for every textual column in the schema in order to maximise performance for those cases where the special characters are not required. Note that the Unicode columns will take up twice as many bytes (two per "character") as the non-Unicode columns (one per character) so you may also need to increase the size of columns that you convert to Unicode.

Chapter 6. Installation of KonaKart Enterprise Extensions

This chapter explains how to install the KonaKart Enterprise Extensions.

Please read this section carefully before attempting to install the Enterprise Extensions.

Before You Begin

The KonaKart Enterprise Extensions are installed "on top of" a standard Community Edition installation of KonaKart. Therefore, before you begin, ensure that you have installed KonaKart Community Edition successfully.

The Enterprise Extensions require a Community Edition installation of the same version. Therefore, if you are using an earlier version of KonaKart, you will need to upgrade to the appropriate Community Edition version before attempting an installation of the Enterprise Extensions. There are database upgrade scripts available to help you migrate up to new versions of KonaKart - see the KonaKart installation for details.

Ensure that KonaKart has been stopped before you start the installation process.

Ensure that you have backed up your existing KonaKart data before proceeding. This is a critical step before any upgrade or installation of KonaKart.

There are no additional platform requirements for the Enterprise Extensions - so if your Community Edition of KonaKart is running successfully on your particular platform, you can have confidence that the Enterprise Extensions will also work.

Licensing

You will need a valid license to run the Enterprise Extensions. You need to obtain a license owner code and a license key from KonaKart then enter this in the Licensing Panel of the KonaKart Administration Application. The key will have an expiry date that will be displayed on the Licensing Panel.

Pre-requisites

The only pre-requisite is:

- A successfully installed KonaKart Community Edition

Create a Database

You will already have created a database for your Community Edition installation which should be working successfully in the standard single-store mode.

Note that when this section of the guide talks about a "database" what it means is a defined set of tables for a defined user and password. Different database suppliers implement this separation differently but so long as you set up your database so that your defined user can create and access his own tables, you can set up your database as you like. In some cases you may wish to load the KonaKart tables in an existing schema for ease of integration to other applications. So long as there are no database object (tables names etc) conflicts, this is fine.

Now you have to consider whether or not you need a new database for the Enterprise Extensions. The KonaKart Engine Mode you require will affect what databases you will have to create.

- Single Store Mode

No new databases are required. The original database set up for your existing Community Edition will be used.

- Multi-Store Shared DB Mode

No new databases are required. All new stores will use the existing Community Edition database.

- Multi-Store Non-Shared DB Mode

A new database ("schema", "user" etc. - see above) is required for every new store. The KonaKart Enterprise Extensions installation only supports the population of a second store (called "store2"). If you need to set up additional stores in this mode you will have to create additional databases for those and configure the konakart properties files manually. This is easy to do by following the examples that the Enterprise Extensions installation creates for "store1" and "store2".

You will need to populate these newly created databases (that is, databases you create after "store2" which is populated by the installer). Populate them using the konakart_demo.sql SQL script which can be found for all supported databases under the database directory under your KonaKart installation directory.

Installing Enterprise Extensions

Use of the KonaKart Enterprise Extensions requires that you first sign an "End User License Agreement". Please see the KonaKart website for further details.

Once you have signed the license agreement you can proceed to install the Enterprise Extensions.

Note that if the GUI or silent installers do not work on your platform you should download the zip version of KonaKart and follow the manual installation instructions.

Take Careful Note: The installation can be configured to execute SQL commands that will change data in your database. Ensure that you have saved a backup of your database before you begin the installation just in case you have to revert.

Installing KonaKart Enterprise Extensions on Windows

Run the set-up program that executes a graphical installation wizard - see Graphical Installation Wizard below. (You can use the "Silent Mode" installation if you prefer, but the graphical version is probably easier if you're installing for the first time).

Installing KonaKart Enterprise Extensions on Unix/Linux

Create a terminal session on your machine and enter the following: (You may prefer to use commands to do the same thing from your X-desktop if you have one installed).

```
$# (replace 3.2.0.0 with the version you wish to install)
$ chmod +x KonaKart-Enterprise-3.2.0.0-Linux-Install
```

If you have a graphical environment on your Linux/Unix machine you will be able to run the GUI. In which case see the Graphical Installation Wizard below (identical steps to the Windows installation).

If you don't have a graphical environment you will see this warning message:

..../src/programlistings/PL_graphical_env.xml

Silent Mode Installations

When running in "silent" (-S) (or "unattended") mode you are able to specify configuration parameters on the command line, for example:

```
$ ./KonaKart-Enterprise-3.2.0.0-Linux-Install -S \
    -DDatabaseType mysql \
    -DSharedCustomers True \
    -DEngineMode 2 \
    -DLoadDB 1 \
    -DJavaJRE "/usr/lib/jvm/java-6-sun/"
```

Note that by default the database tables are not created or populated. If you want the tables created and populated you need to set the LoadDB parameter to 1.

Silent Mode Parameters

The following parameters can be added to the command line, as in the example above, to specify default values for KonaKart at installation time:

Note that the database user, password and URL are only required when you need to define a new database (for example when creating a Multi-Store Multi-Database installation - which is Engine Mode 1).

Parameter	Default Value	Explanation
DatabaseType	mysql	mysql, postgresql, db2net, oracle, mssql
DatabaseUrl	jdbc:mysql://localhost:3306/db-name?zeroDateTimeBehavior=convertToNull	Database URL
DatabaseUsername	root	Database User's Username
DatabasePassword		Database User's Password
DatabaseDriver	com.mysql.jdbc.Driver	Database Driver
mssqlDBO	dbo	Database Owner (only used by MS SQL Server)
InstallationDir	Windows: C:\Program Files\KonaKart *nix (as root): /usr/local *nix (as user): ~/konakart	Installation Directory
LoadDB	0	1=Load DB 0=Do not Load DB
JavaJRE		The Java runtime location
EngineMode	0	KonaKart Engine Mode (0,1 or 2)

SharedCustomers	True	Share Customers (true or false)
SharedProducts	False	Share Products (true or false)
SharedCategories	False	Share Categories (true or false)

Note that currently you can only have shared Products if you also share Customers and you can only have shared Categories if you have shared Products.

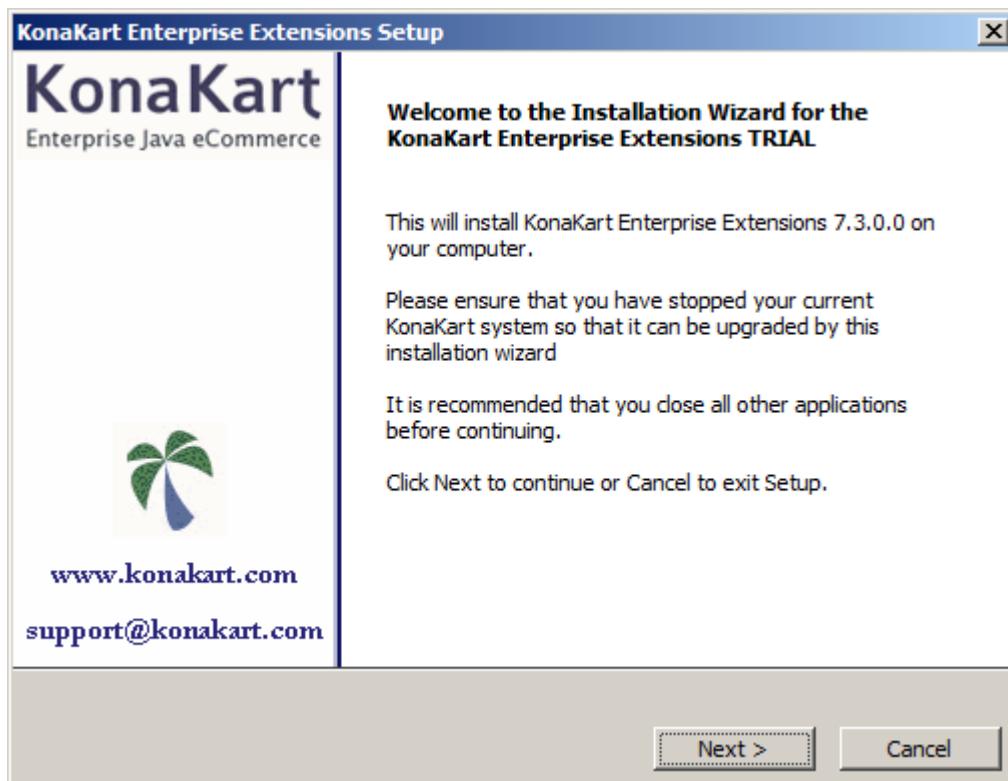
Graphical Installation Wizard

This shows a typical installation that uses the wizard:

Either double-click on the installation setup program (either KonaKart-Enterprise-3.2.0.0-Windows-Setup.exe on Windows, or KonaKart-Enterprise-3.2.0.0-Linux-Install on Linux - or respective later version numbers) or run it from a command shell. You are first presented with this small window which allows you to confirm that you wish to proceed with the installation:

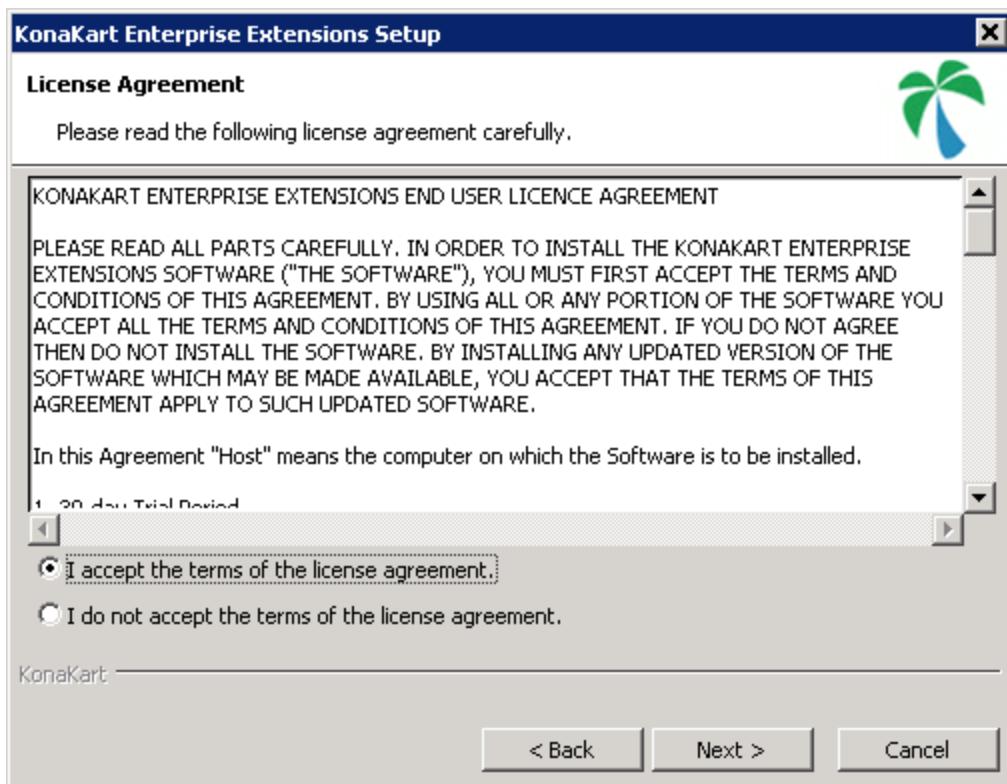


Click on Yes to continue to:



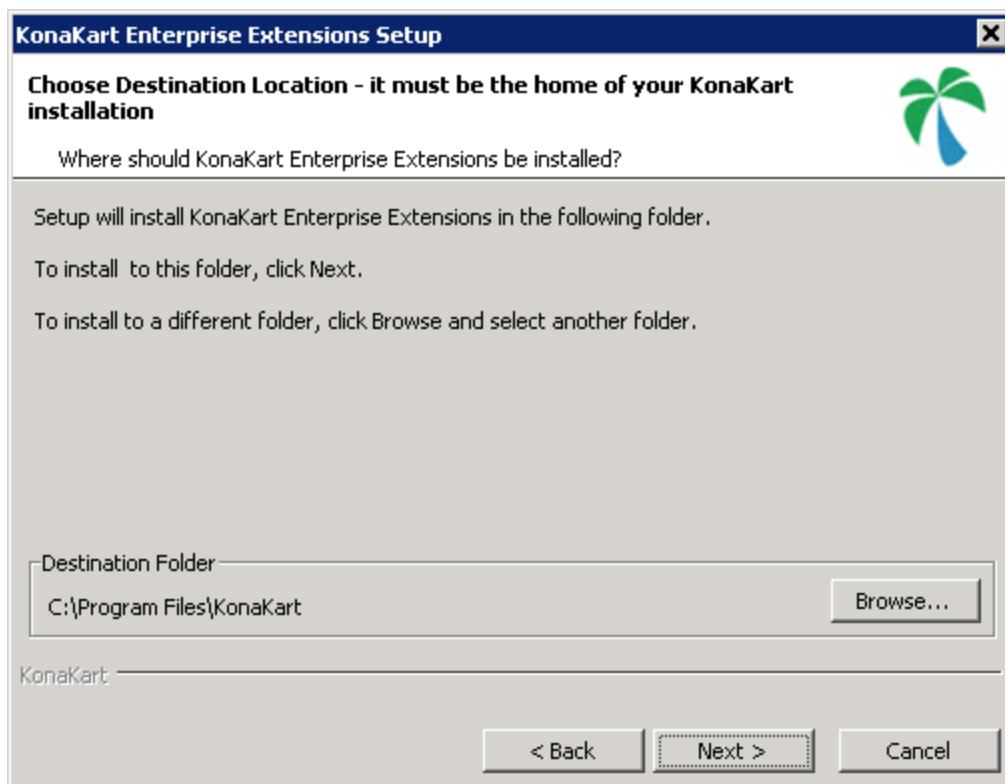
Installation of KonaKart
Enterprise Extensions

Check that you have the correct version number and click on next to get to the next screen. Note in passing the email address for support questions. Please contact us and / or search the forum if you have any difficulties at all with the installation and we will endeavour to help you as soon as possible.



Please read the license agreement carefully and if you are happy to do so under the terms of the agreement, click on the "I accept the terms of the license agreement" bullet and click next to continue. If you are not prepared to accept that license agreement please quit the installation at this point and delete all copies of the software.

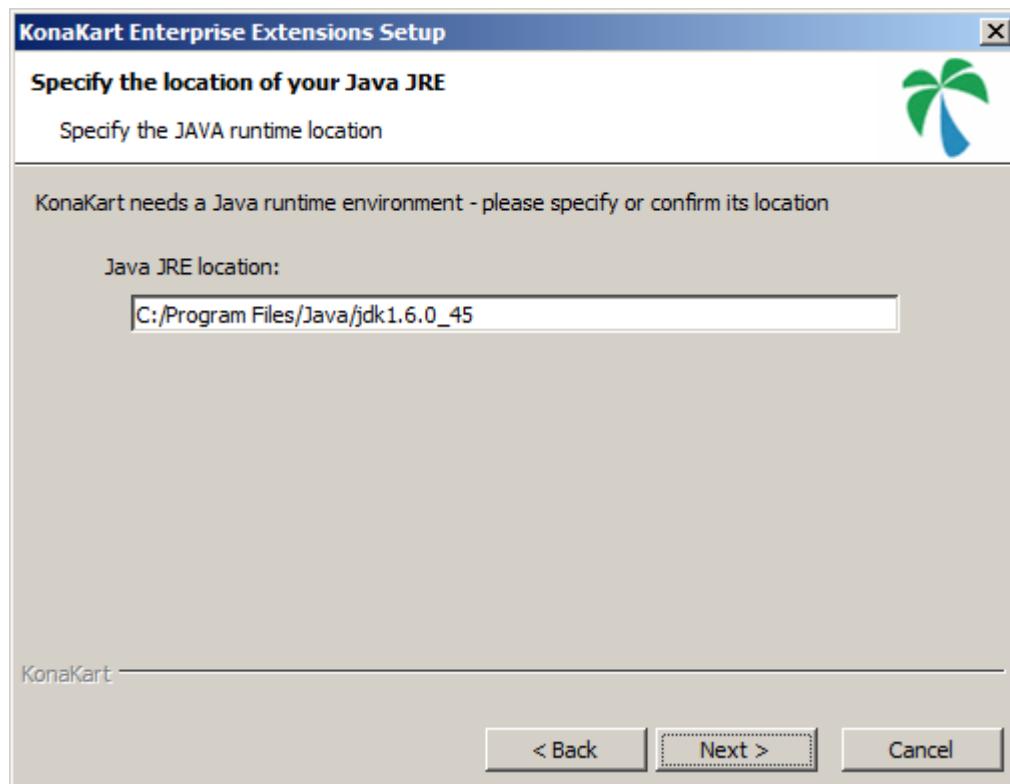
Click on next to reach:



This is where you specify where you previously installed the Community Edition of KonaKart. On Windows this defaults to "C:\Program Files\", on Linux this is the user's home directory (if the user is not root) or "/usr/local" (if the user is root). This can be specified in the silent mode of on the command line of the GUI version using -DInstallationDir.

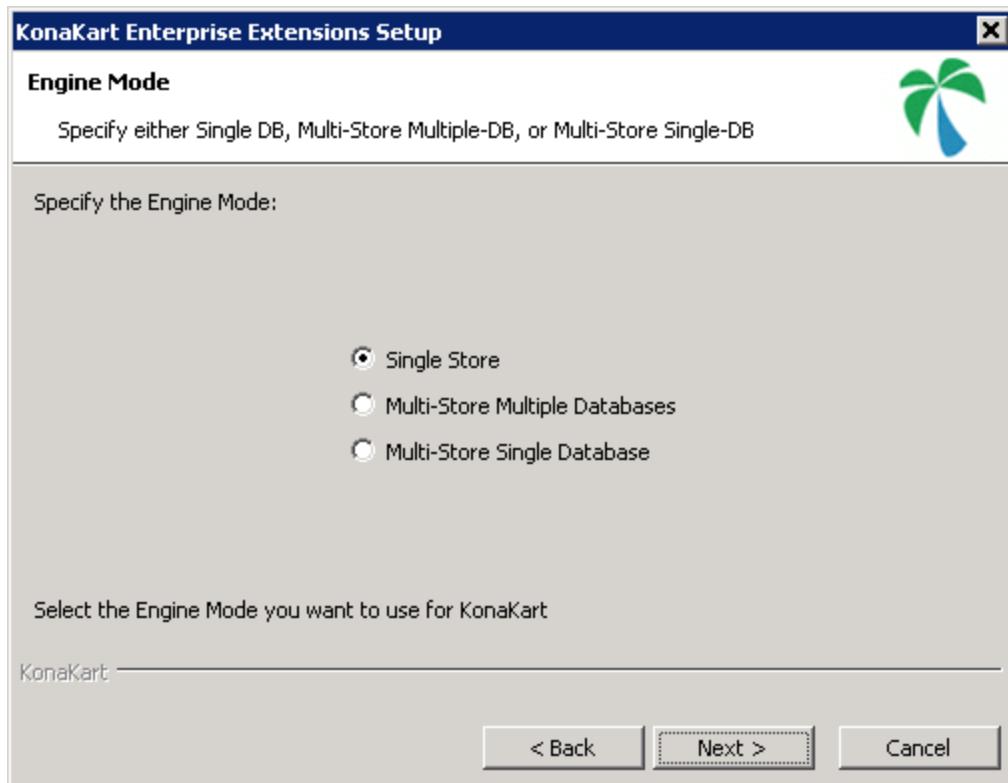
The location specified must match the location of your Community Edition installation.

On clicking next you reach:



Here you have to confirm or specify the location of the java runtime environment. The wizard will try to find this for you but it is not always successful. In the cases where it isn't successful you will have to enter the location manually. If you have installed java in the default location or it appears in your environment's path, the wizard should find it for you.

Click on next to get to:



This is where you define KonaKart Engine Mode. There are three choices here:

- Single Store (Engine Mode 0)

This is similar to the way the Community Edition works. There is one database with one store.

- Multi-Store Multiple Databases (Engine Mode 1)

In this mode, one instance of KonaKart manages multiple stores with the data for each store being held in a separate database. This mode offers some security advantages (database credentials are not shared between stores) but is more effort to maintain as you cannot dynamically-create new stores.

- Multi-Store Single Database (Engine Mode 2)

In Engine Mode 2 one KonaKart instance manages multiple stores in one database. This is a flexible mode that allows the creation of new stores dynamically. Within this mode there are additional options to define which KonaKart objects (products/customers) you wish to "share" between stores:

- Shared Products Mode

In this mode products may be shared amongst the different stores in the KonaKart "mall". Therefore, a product can be made available for sale in any subset of the stores in the mall in Shared Products Mode. To use Shared Products you must also have Shared Customers (see below).

If some subset of the products are shared between stores, this mode can be advantageous because there is only one copy of each product so maintenance of products can be reduced. If you update the description for a shared product, this description will be available in all stores that share that product.

- Non-Shared Products Mode

In this mode, the products are not sharable between the individual stores in the mall. The products must be maintained independently for each store. In Non-Shared Products Mode it's possible to have Shared Customers OR Non-Shared Customers (see below).

- Shared Categories Mode

In this mode categories may be shared amongst the different stores in the KonaKart "mall". To use Shared Categories you must also have Shared Products (which in turn also requires Shared Customers (see below)).

If categories are shared between stores, this mode can be advantageous because there is only one copy of each category so maintenance can be reduced.

- Non-Shared Categories Mode

In this mode, the categories are not sharable between the individual stores in the mall. The categories must be maintained independently for each store but it allows different categories for each store. In Non-Shared Categories Mode it's possible to have Shared Products OR Non-Shared Products.

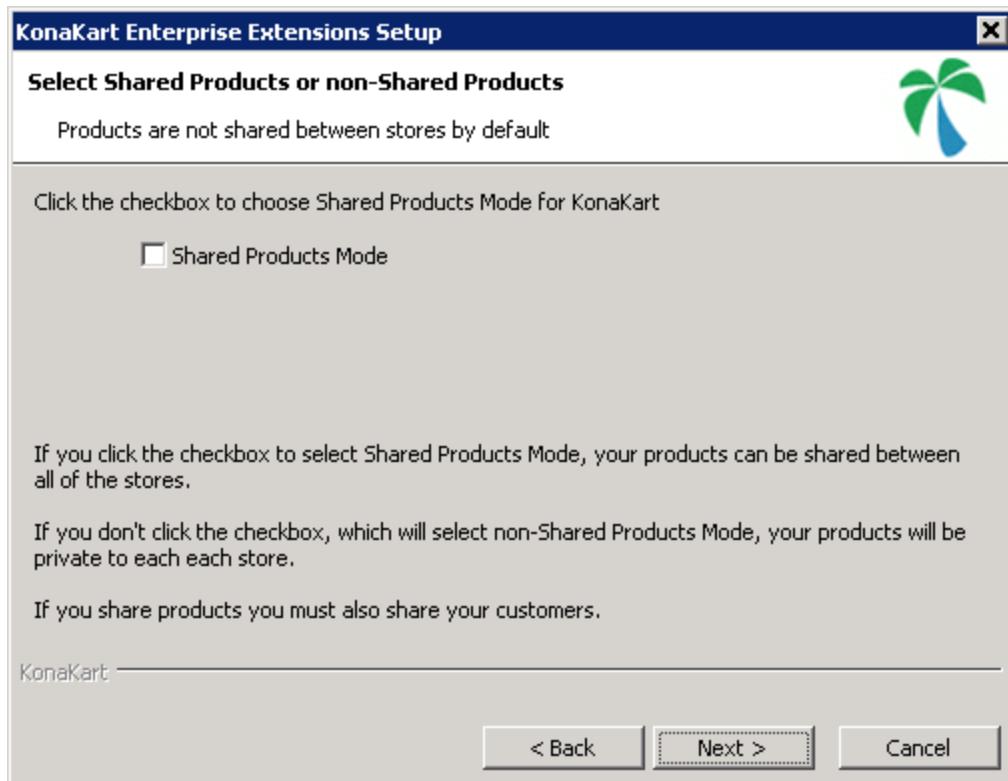
- Shared Customers Mode

In this mode customers are shared amongst the different stores in the KonaKart "mall". Therefore, a customer who has registered an account with one of the stores in the KonaKart instance, can use the same credentials to log on to any other store managed by the KonaKart instance.

- Non-Shared Customers Mode

In this mode, one instance of KonaKart manages multiple stores with the data for each store being held in a separate database. This mode offers some security advantages (database credentials are not shared between stores) but, if you need to create a large number of stores, requires more effort to maintain as you cannot dynamically-create new stores.

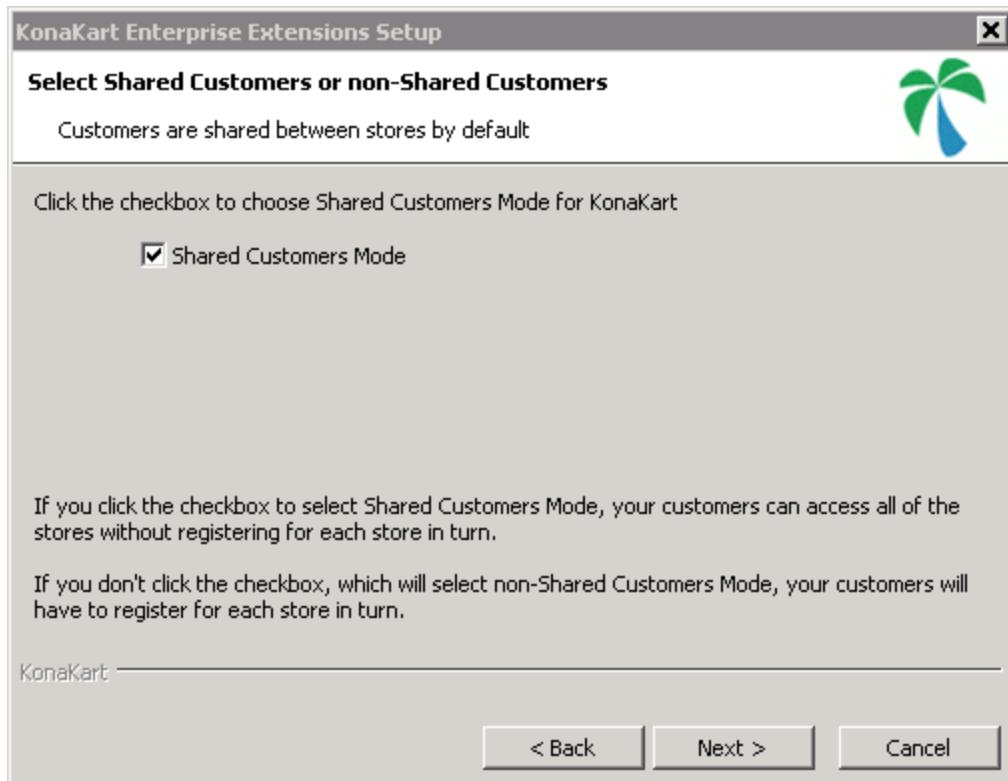
The next screen shown is dependent on which Engine Mode you select. If you select Multi-Store Single DB, you will be presented with this screen, offering you the choice of having "Shared Products" or "Non-Shared Products":



Click the checkbox to choose Shared Products Mode which will allow you to share products between the multiple stores in your mall.

When you click on the "Next" button the next screen presented to you will depend on whether you chose Shared Products or not. Since with Shared Products you must also have Shared Customers Mode, if Shared Products is selected, you will then skip the Shared Customers Choice screen and be back on the same path as if you'd chosen one of the other Engine Modes.

However, if you chose Non-Shared Products mode, you are presented with the following screen where you choose between Shared and Non-Shared Customers:

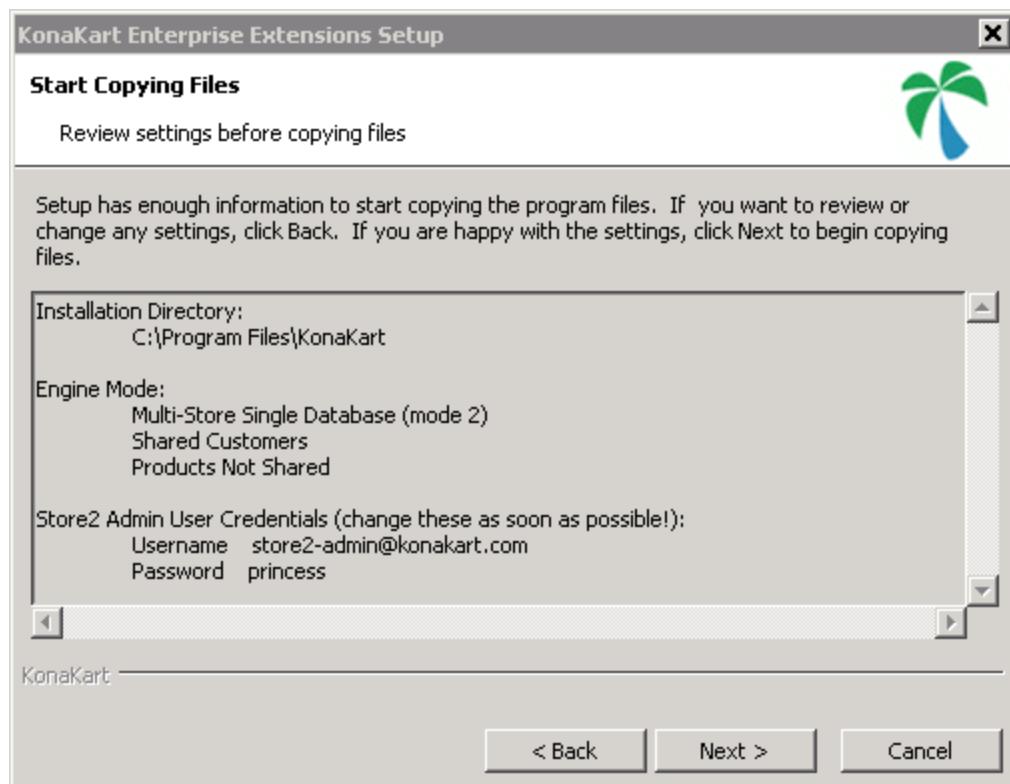


Click the checkbox to choose Shared Customers Mode which will allow customers who register in one of the stores in your KonaKart instance, to be able to log in to the other stores in the KonaKart instance using the same credentials.

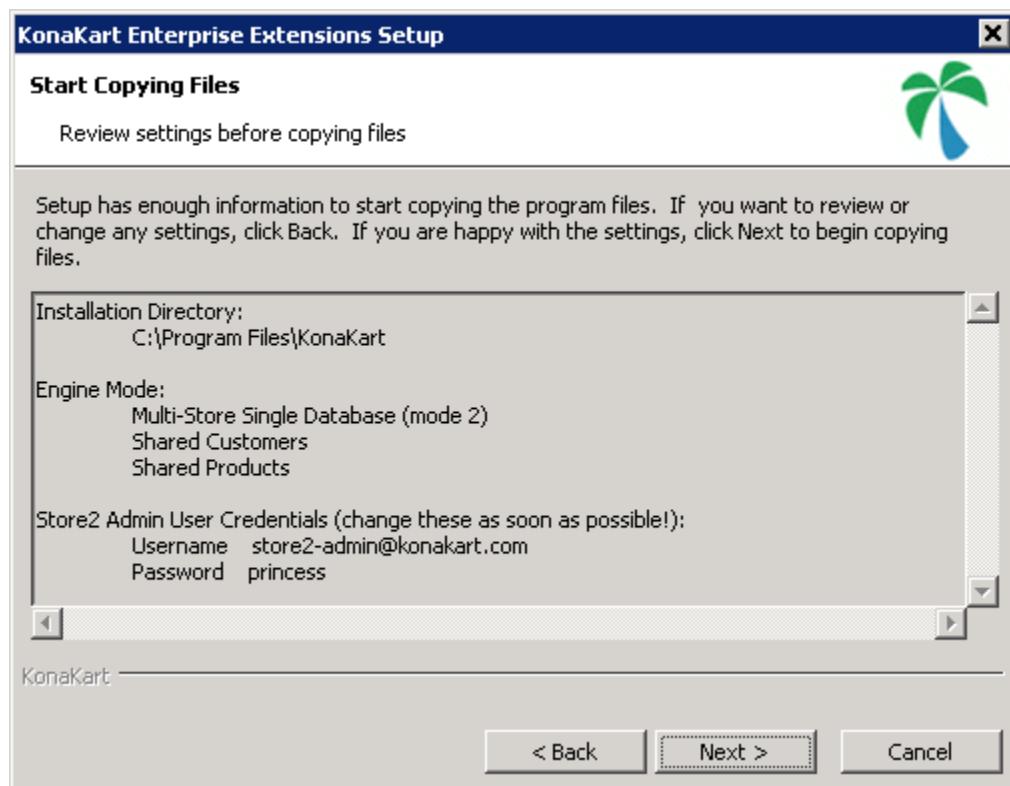
When you click on the "Next" button you will be back on the same path as if you'd chosen one of the other Engine Modes on an earlier screen in the wizard.

If you had selected not to Share Products but to Share Customers the next screen will appear as follows:

Installation of KonaKart
Enterprise Extensions



If you had selected Shared Products Mode and Shared Customers Mode this screen would appear as follows:



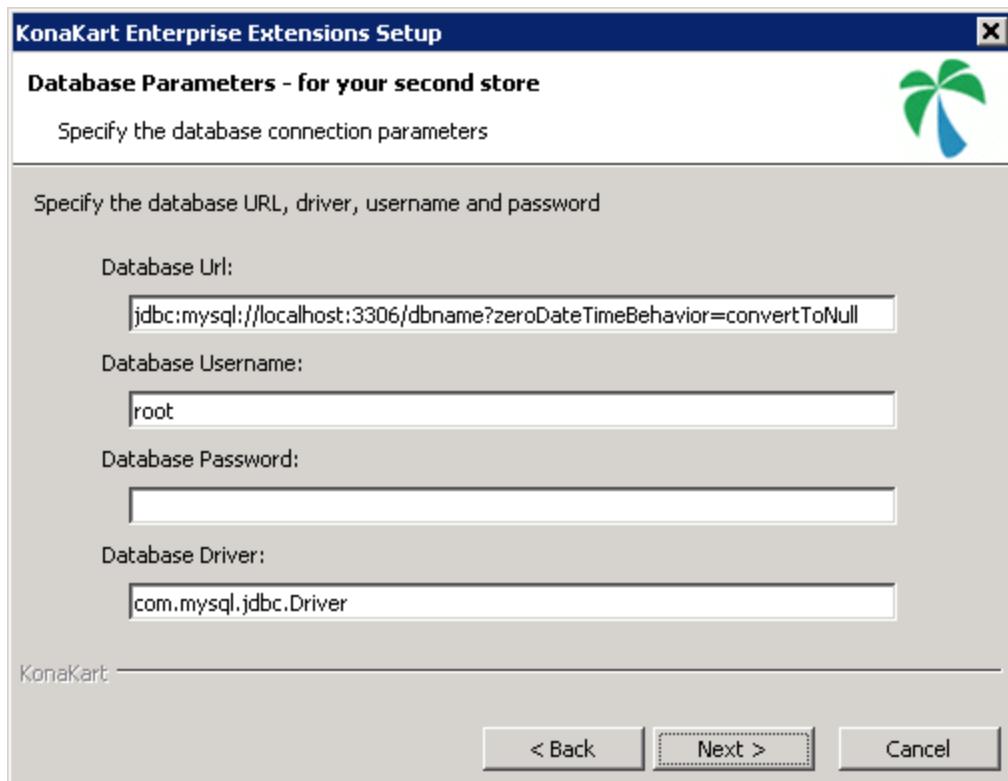
This is your final chance to check your settings before copying the files into position. (The values shown on the screen are dependent on the settings you have selected earlier in the wizard).

Clicking next will start the process of copying the KonaKart Enterprise Edition files into position. The copying of the files is shown on an "Installing Files..." window but there aren't very many files so you may find that it only flashes up for a second or two before moving forward automatically to the next screen.

The next screen in the wizard is again dependent on the selected Engine Mode.

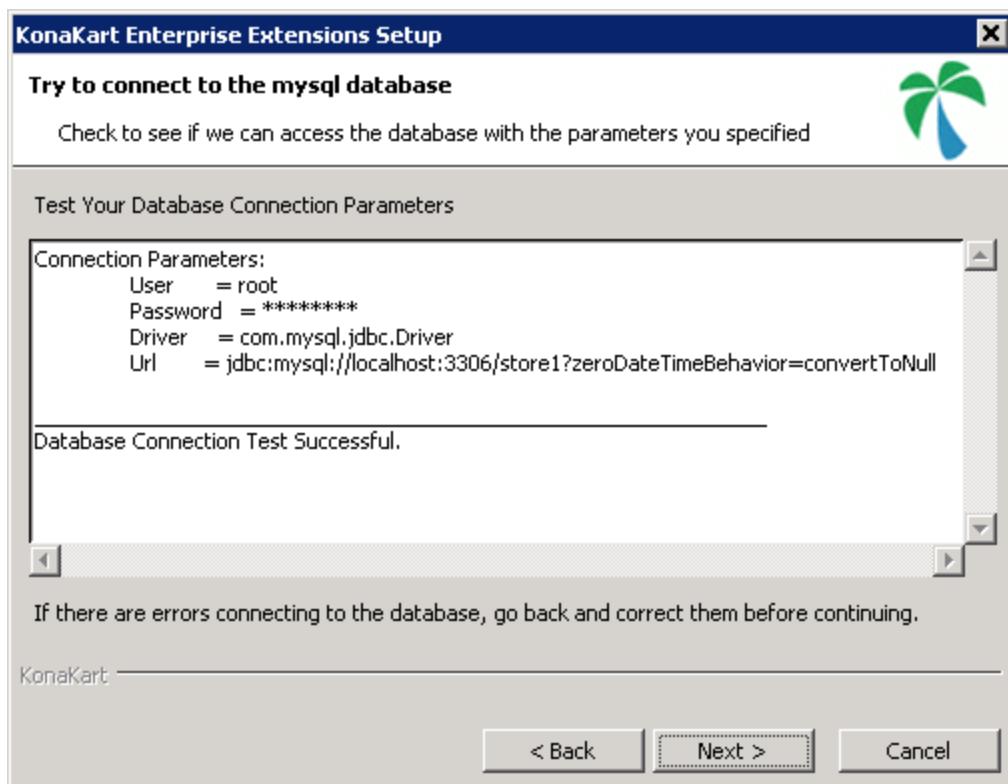
For Single Store Mode you are taken to the very last screen in the wizard which tells you whether or not the installation has been successful and gives you the option to restart KonaKart with the newly installed Enterprise Extensions.

For Multi-Store Multiple DB Mode you are taken to a screen that asks for the database connection parameters for your second store:

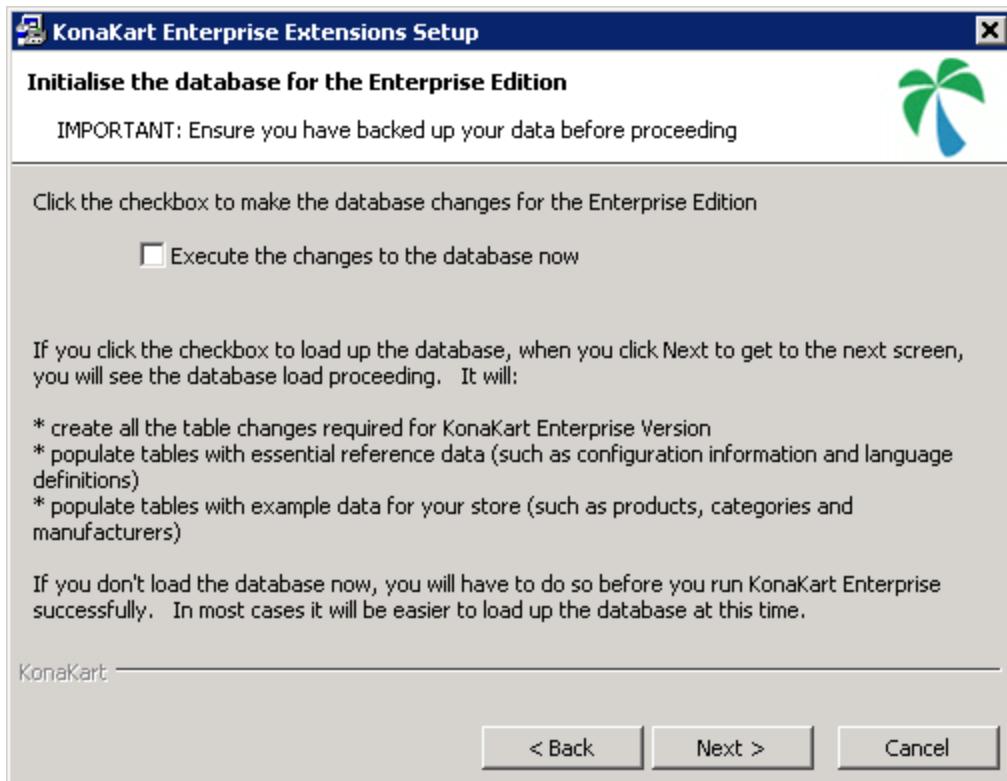


After setting these parameters and clicking on the "Next" button you are taken to a database connection test screen (as below for the Multi-Store Single Database mode)

For Multi-Store Single DB Mode you are taken to a Database Connection Test window which will check to see whether or not it can connect to the database that your existing Community Edition installation of KonaKart is using. This screen looks like this after a successful connection test:



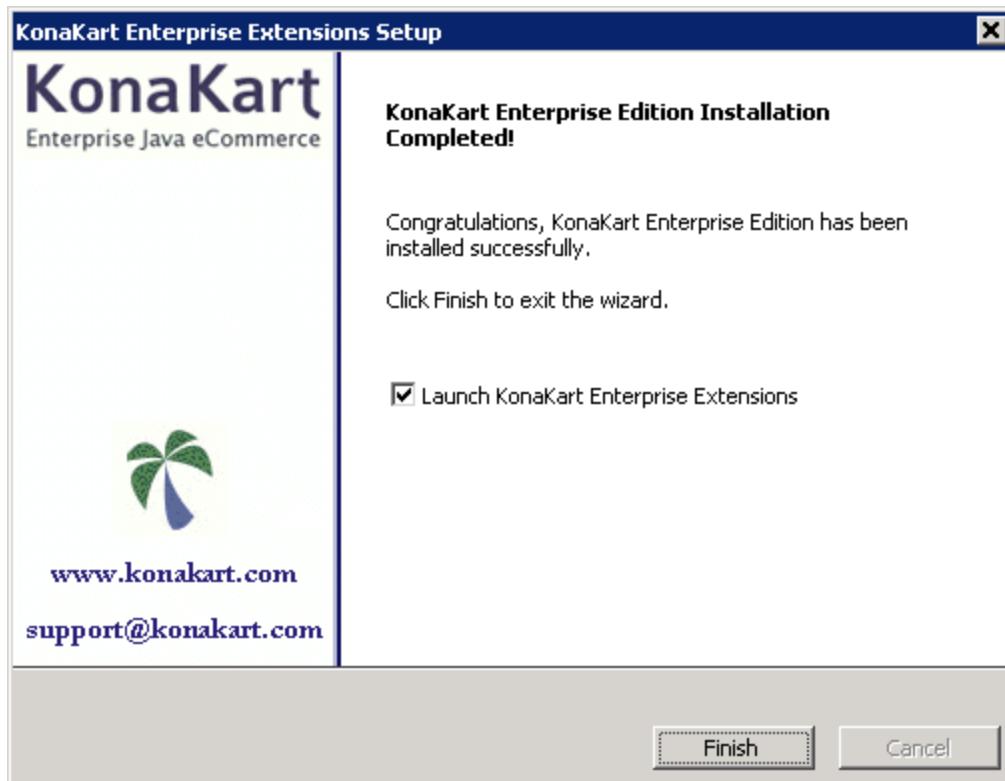
After a successful database connection test, you are asked whether or not you want to load the database with the new Multi-Store data. In most cases you will want to load the data, so click the checkbox on this screen:



After about 10 secs (Single DB mode) or about 60 secs (Multiple DB mode) you will be presented with a screen that will report the success or failure of the database load. (Naturally, the time it takes to complete the database load will depend on how fast your system operates).

If there's a problem with the database load, double-check these installation instructions, check the KonaKart forum for similar problems or, if you have a support contract, contact the KonaKart support team for assistance.

After a successful installation of the KonaKart Enterprise Extensions you will be presented with the following screen:



If you leave the checkbox as it is (the default is checked) the wizard will re-start KonaKart with the Enterprise Extensions, running in the Engine Mode you have chosen.

Click "Finish" to finish the installation.

Run KonaKart and KonaKartAdmin exactly as you did with the Community Edition by entering:

<http://localhost:8780/konakart/> [<http://localhost:8780/konakart/>] or <http://localhost:8780/konakartadmin/> [<http://localhost:8780/konakartadmin/>] in your browser, adjusting the port as necessary if you deployed to a different port number.

Manual Installation of the Enterprise Extensions

If you are installing on a platform that supports the GUI installer (Windows, Linux, Unix), it is recommended that you use that. If not, but you are on a platform that supports the silent form of the installer (again Windows, Linux, Unix), it is recommended that you use that. Otherwise, or if you have other requirements, use the manual installation.

The installation work required is dependent on your target environment. Follow the guidelines for manual installation for your target platform that are documented for the Community Edition. These instructions will supplement those.

However you plan to install KonaKart Enterprise Extensions, it is still advisable to run through the GUI installer if you can. The reason for this is that it will populate all the properties files for you and load your database automatically. Once you have done this you can make WARs from the GUI-installed version of KonaKart (details below) and deploy them elsewhere as you please.

The KonaKart Enterprise Extensions Installation section contains general KonaKart Enterprise Extensions installation instructions (although focuses on using the GUI-driven and silent versions of the installer) and contains important information that is also relevant for the manual installation so check this before starting out.

In general, you need to follow all the documented installation instructions except for the "Install Enterprise Extensions" section which explains how to use the automated GUI and Silent versions of the installation.

Perform all the documented installation instructions for:

- A Java runtime environment
- A database loaded with KonaKart tables
- A successfully working KonaKart Community Edition

For the purposes of this guide we'll use MySQL and a database named "store1" (and "store2" where applicable).

1. Copy the Enterprise Extensions files into position

Unzip the KonaKart-Enterprise-n.n.n.n.zip file for your version on top of the existing KonaKart Community edition installation. Ensure that you unzip to the home directory of the existing KonaKart installation so that all the files are loaded into the correct positions.

2. Modify the KonaKart Configuration Files

a. Set Database Parameters

Modifications should only be required if you have chosen Multi-Store Multiple Database Mode (Engine Mode 1). For all other modes, the database parameters should already be set correctly.

Set DB name (and other database connection parameters) in *konakart.properties* and *konakartadmin.properties*. Change the string "dbname" in the URL for MySQL to the name of your database (in this case "konakart") in the following files:

{konakart}/webapps/konakart/WEB-INF/classes/konakart.properties

{konakart}/webapps/konakartadmin/WEB-INF/classes/konakartadmin.properties

This is documented in the "Defining the Database Parameters" section of the Community Edition installation.

For Multi-Store Single DB Modes (Engine Mode 2) there is no difference to the database definition required in the two properties files. However, if you are using the Multi-Store Multiple DBs Mode (Engine Mode 1) you will need to enter database connection credentials for each of your stores. An example of a two-store set-up is:

```

# -----
# D A T A B A S E   P R O P E R T I E S
# -----


torque.applicationRoot = .

torque.database.default = store1

torque.database.store1.adapter = mysql
torque.dsfactory.store1.connection.driver = com.mysql.jdbc.Driver
torque.dsfactory.store1.connection.url =
    jdbc:mysql://localhost:3306/store?zeroDateTimeBehavior=convertToNull
torque.dsfactory.store1.connection.user = root
torque.dsfactory.store1.connection.password =


# Enterprise Feature
torque.database.store2.adapter = mysql
torque.dsfactory.store2.connection.driver = com.mysql.jdbc.Driver
torque.dsfactory.store2.connection.url =
    jdbc:mysql://localhost:3306/store2?zeroDateTimeBehavior=convertToNull
torque.dsfactory.store2.connection.user = root
torque.dsfactory.store2.connection.password =


# -----
# C O N N E C T I O N   P O O L   P R O P E R T I E S
# -----
# We can leave the defaults
# -----


# Using commons-dbcp

torque.dsfactory.store1.factory=org.apache.torque.dsfactory.SharedPoolDataSourceFactory
torque.dsfactory.store2.factory=org.apache.torque.dsfactory.SharedPoolDataSourceFactory

# The maximum number of active connections that can be allocated from this pool at
# the same time, or zero for no limit.

torque.dsfactory.store1.pool.maxActive=0
torque.dsfactory.store2.pool.maxActive=0

# The maximum number of active connections that can remain idle in the pool, without
# extra ones being released, or zero for no limit.

torque.dsfactory.store1.pool.maxIdle=10
torque.dsfactory.store2.pool.maxIdle=10

# The maximum number of milliseconds that the pool will wait (when there are no
# available connections) for a connection to be returned before throwing an exception,
# or -1 to wait indefinitely.

torque.dsfactory.store1.pool.maxWait=-1
torque.dsfactory.store2.pool.maxWait=-1

# The indication of whether objects will be validated before being borrowed from the
# pool. If the object fails to validate, it will be dropped from the pool, and we will
# attempt to borrow another.

torque.dsfactory.store1.pool.testOnBorrow=true
torque.dsfactory.store2.pool.testOnBorrow=true

# The SQL query that will be used to validate connections from this pool before
# returning them to the caller. If specified, this query MUST be an SQL SELECT
# statement that returns at least one row.
# Recommended settings:
# for MySQL/PostgreSQL/MS SQL use: SELECT 1
# for Oracle           use: SELECT 1 from dual
# for DB2              use: SELECT 1 FROM sysibm.sysdummy1

torque.dsfactory.store1.pool.validationQuery=SELECT 1
torque.dsfactory.store2.pool.validationQuery=SELECT 1

```

Notice how most of the Torque parameters are repeated for each store.

Configure the "EE" variants of the managers in konakart.properties as required:

```
konakart.manager.ProductMgr = com.konakart.bl.ProductMgrEE
konakart.manager.SecurityMgr = com.konakart.bl.SecurityMgrEE
konakart.manager.OrderMgr = com.konakart.bl.OrderMgrEE
```

b. konakartadmin.properties

The changes required are dependent on the chosen Engine Mode.

For Multi-Store Multiple Databases Mode (Engine Mode 1) you need to define the databases that are used for each store. These parameters can be ignored for other Engine Modes. This is an example of a two database setup with databases store1 and store2:

```
# -----
# Enterprise Feature
# The databases actually used in a multi store / multi database environment
# The "used" database definitions must map to the Torque definitions above
# The "description.*" definitions are friendly names for the Stores

konakart.databases.used = store1 store2
konakart.databases.description.store1 = Store1
konakart.databases.description.store2 = Store2
```

For all Multi-Store Modes (Engine Modes 1 and 2) you need to define the Engine Mode that the web services engines will use: This is an example of a definition for Engine Mode 2:

```
# -----
# Enterprise Feature
# Engine mode that the web services engine will use
# 0 = Single Store (default)
# 1 = Multi-Store Multiple-Databases (add konakart.databases.used above as well)
# 2 = Multi-Store Single Database

konakart.ws.mode = 2
```

Note the reminder in the comment above to set the *konakart.databases.used* property for Engine Mode 1 (Multi-Store Multiple Database Mode).

For Multi-Store Single Database Mode (Engine Mode 2) you need to define whether or not you are operating in the Shared Customers Mode. This setting is not used for other engine modes. This is an example of a definition for defining that you wish to use Shared Customers:

```
# -----
# Enterprise Feature
# When in multi-store single database mode, the customers can be shared between stores

konakart.ws.customersShared = true
```

For Multi-Store Single Database Mode (Engine Mode 2) you also need to define whether or not you are operating in the Shared Products Mode. This setting is not used for other engine modes. This is an example of a definition for defining that you wish to use Shared Products:

```
# -----
# Enterprise Feature
# When in multi-store single database mode, the products can be shared between stores
konakart.ws.productsShared = true
```

Set the "EE" variants of the managers in konakartadmin.properties as required:

```
konakart.admin_manager.AdminSecurityMgr = com.konakartadmin.bl.AdminSecurityMgrEE
```

c. konakartadmin_gwt.properties

Modify this file as required to set the Engine Mode and appropriate value for customersShared and productsShared. The file is deployed to *{konakart}/webapps/konakartadmin/WEB-INF/classes/* .

The following example shows a setup for EngineMode 2 with Shared Customers and Shared Products:

```
# -----
# Enterprise Feature
# Engine mode that the KonaKart Admin engine will use
# 0 = Single Store (default)
# 1 = Multi-Store Multiple-Databases (add konakart.databases.used above as well)
# 2 = Multi-Store Single Database

konakartadmin.gwt.mode = 2

# -----
# Enterprise Feature
# When in multi-store single database mode, the customers can be shared between stores
konakartadmin.gwt.customersShared = true

# When in multi-store single database mode, the products can be shared between stores
konakartadmin.gwt.productsShared = true
```

d. konakart.properties

The changes required are dependent on the chosen Engine Mode.

The database parameters should be set up in the same way as you did in the konakartadmin.properties file (see above).

For Multi-Store Multiple Databases Mode (Engine Mode 1) you need to define the databases that are used for each store. These parameters can be ignored for other Engine Modes. This is an example of a two database setup with databases store1 and store2:

```
# Enterprise Feature
# The databases actually used in a multi store / multi database environment
konakart.databases.used = store1 store2
```

Define the web services Engine Mode and the customers shared property as you did in konakartadmin.properties (see above). Here is an example of a setup that defines Engine Mode 2 without sharing customers or products:

```
# -----
# Enterprise Feature
# Engine mode that the web services engine will use
# 0 = Single Store (default)
# 1 = Multi-Store Multiple-Databases (add konakart.databases.used above as well)
# 2 = Multi-Store Single Database

konakart.ws.mode = 2

# -----
# Enterprise Feature
# When in multi-store single database mode, the customers can be shared between stores

konakart.ws.customersShared = false

# When in multi-store single database mode, the products can be shared between stores

konakart.ws.productsShared = false
```

e. konakart and konakart-m web.xml files

A few modifications need to be made to the web.xml files that use Struts to set the mode of the KonaKart plugins. The Engine Mode must be set to 0 (Single Store mode), 1 (Multi-Store Multiple DBs Mode) or 2 (Multi-Store Single DB Mode). If you are using Engine Mode 2 and you have chosen Shared Customers you must also set customersShared to "true" otherwise leave that as "false". Likewise, set productsShared and categories Shared to true or false as appropriate.

The konakart-m webapp was last included in the v6.2.0.2 release so this next section is only relevant if you have the konakart-m webapp present.

The following is the setting for the konakart-m webapp's web.xml:

```
<!-- Start a KKAppEngine -->
<servlet>
  <description>KonaKart Mobile Client Engine Servlet</description>
  <servlet-name>KonaKartAppEngineServlet</servlet-name>
  <servlet-class>com.konakart.servlet.AppEngServlet</servlet-class>
  <init-param>
    <param-name>propertiesPath</param-name>
    <param-value>konakart.properties</param-value>
  </init-param>
  <init-param>
    <param-name>appPropertiesPath</param-name>
    <param-value>konakart_app.properties</param-value>
  </init-param>
  <init-param>
    <param-name>mode</param-name>
    <param-value>2</param-value>
  </init-param>
  <init-param>
    <param-name>storeId</param-name>
    <param-value>store1</param-value>
  </init-param>
  <init-param>
    <param-name>customersShared</param-name>
    <param-value>true</param-value>
  </init-param>
  <init-param>
    <param-name>productsShared</param-name>
    <param-value>true</param-value>
  </init-param>
  <init-param>
    <param-name>categoriesShared</param-name>
    <param-value>true</param-value>
  </init-param>
  <init-param>
    <param-name>defaultstoreId</param-name>
    <param-value>store1</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

A very similar servlet definition needs to be amended for the konakart web app as well.

f. Populate the Database ready for the Enterprise Extensions

This is an example of a Windows BAT file that you should run for setting up a Multi-Store Single DB (Engine Mode 2) system. This utility also creates a second sample store (called "store2") just as is done in the automated installation process. You need to configure the properties files (see above) before executing this program because it needs to know the chosen configuration:

```

@echo off
rem
rem Set up database for KonaKart Enterprise Extensions
rem Also sets up a second sample store (store2) in the database
rem - Manual Installation Only
rem
rem Normally this is executed by the Enterprise Extensions Installation Wizard.
rem
rem Always use the Enterprise Extensions Installation Wizard in preference to
rem running this manually.
rem

set INSTALL_DIR=C:/Program Files/KonaKart
set WEBAPPS_DIR=C:/Program Files/KonaKart/webapps
set DBDIR=MySQL
set SHARED_CUSTOMERS=True
set SHARED_PRODUCTS=True
set SHARED_CATS=True

set CP=%WEBAPPS_DIR%\konakartadmin\WEB-INF\classes
set CP=%CP%;%WEBAPPS_DIR%\konakartadmin\WEB-INF\lib\konakartadmin_multistore.jar
set CP=%CP%;%WEBAPPS_DIR%\konakartadmin\WEB-INF\lib\konakart_custom_utils.jar
set CP=%CP%;%WEBAPPS_DIR%\konakartadmin\WEB-INF\lib\konakartadmin.jar
set CP=%CP%;%WEBAPPS_DIR%\konakartadmin\WEB-INF\lib\konakartadmin_multistore.jar
set CP=%CP%;%WEBAPPS_DIR%\konakartadmin\WEB-INF\lib\konakartadmin_enterprise.jar
set CP=%CP%;%WEBAPPS_DIR%\konakartadmin\WEB-INF\lib\konakartadmin_solr.jar
set CP=%CP%;%WEBAPPS_DIR%\konakartadmin\WEB-INF\lib\konakartadmin_publishproducts.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\konakart.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\konakart_utils.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\konakart_torque-4.0.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\velocity-1.5.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\konakart_village-4.0.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\commons-pool-1.3.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\commons-dbcp-1.4.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\commons-configuration-1.7.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\commons-beanutils-1.8.0.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\commons-lang-2.4.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\commons-collections-3.2.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\commons-logging-1.1.1.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\log4j-1.2.12.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\mysql-connector-java-5.1.23-bin.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\ojdbc6.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\postgresql-9.1-901.jdbc4.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\jtds-1.2.5.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\db2jcc.jar
set CP=%CP%;%WEBAPPS_DIR%\konakart\WEB-INF\lib\db2jcc_license_cu.jar

"%JAVA_HOME%\bin\java" -cp "%CP%" ^
com.konakartadmin.utils.CreateEnterpriseDB ^
-p "%WEBAPPS_DIR%\konakartadmin\WEB-INF\classes\konakartadmin.properties" ^
-h "%INSTALL_DIR%" ^
-db %DBDIR%
-ps %SHARED_PRODUCTS%
-cs %SHARED_CATS%
-c %SHARED_CUSTOMERS%

rem
rem The "-d" parameter can be added to enable debugging. This is useful if you
rem have problems.
rem

```

Users created by the Enterprise Extensions Installation

This section describes the users that are created by default during the standard KonaKart installations. For more details on creating new stores and defining new users for these new stores, see the section in this User Guide on Multi-Store Configuration .

When you install the Community Edition, the following users are automatically created:

Three users are created with different roles assigned.

Username	Password	Roles
admin@konakart.com	princess	KonaKart Super-User
doe@konakart.com	password	Sample User
cat@konakart.com	princess	KonaKart Catalog Maintainer
order@konakart.com	princess	KonaKart Order and Customer Manager

The above users remain after installing the KonaKart Enterprise Extensions but additional users are created for use with the second store (if one of the Multi-Store Modes is chosen). The above users remain with the same privileges they had originally and can log in to both the Application and the Admin App for store1 as before.

In addition, if and only if, the Multi-Store Single Database with Shared Customers Mode is selected, these users will be able to log on to the Applications for other stores managed by the KonaKart instance. Note that only the Super User user above (`admin@konakart.com`) is granted privileges to log on to the Admin App for the other stores. This is the standard behavior when you opt to allow Shared Customers.

Single Store Mode

No new users are created.

Multi-Store Multiple DB Mode

If Multi-Store Multiple Databases Mode is selected, the users created in store2 match those for store1 since the same data load is performed. In case you need to execute it manually, the SQL file executed is called `konakart_demo.sql` and can be modified (carefully!) to suit local requirements. A `konakart_demo.sql` SQL script file is provided for each of the supported databases under the `database` directory under your KonaKart home directory. In Multi-Store Multiple Databases Mode there is no support for shared customers so the users created are only authorized to log in to their own stores.

Multi-Store Single DB Mode with Shared Customers

If Multi-Store Single Database Mode with Shared Customers is selected, the following user is created:

One user is created with the Store Administrator's role.

Username	Password	Roles
store2-admin@konakart.com	princess	KonaKart Store Administrator

By default the "Store Administrator" role allows the created user to administer the more business-related aspects of a specific store, but not the configuration settings. By "business-related" it means the store administrator user can maintain the products, manufacturers, categories (etc) and administer orders. The "Store Administrator" can log on to the application of any the stores managed by the KonaKart instance, but is not granted any privileges to administer any store other than the one assigned (which is store2 in this case).

In this mode there is no need to create a new "Super User" user as the original "Super User" created for the Community Edition, called `admin@konakart.com`, has sufficient privileges to Administer all other stores in a Shared Customer environment. Note the power of the "Super User"! Ensure that you change

all the user passwords as soon as possible and deny access to the "Super User" account to all who should not be able to use it.

Multi-Store Single DB Mode NON-Shared Customers

If Multi-Store Single Database Mode without Shared Customers is selected, the following users are created by the installer:

One user is created with the Store Administrator's role.

Username	Password	Roles
store2-admin@konakart.com	princess	KonaKart Store Administrator
store2-super@konakart.com	princess	KonaKart Store Super User
store2-doe@konakart.com	password	Sample User
store2-cat@konakart.com	princess	KonaKart Catalog Maintainer
store2-order@konakart.com	princess	KonaKart Order and Customer Manager

By default the "Store Administrator" role allows the created user to administer the more business-related aspects of a store, but not the configuration settings. By "business-related" it means the store administrator user can maintain the products, manufacturers, categories (etc) and administer orders. This user can administer only the store he has been assigned and no others, because he isn't a "Super User".

In this non-customer shared mode, the "Store Administrator" cannot log in to the application of any other store (than the one that he is assigned to) managed by the KonaKart instance because the users aren't shared.

Also, in this mode, a new "Super User" user (*store2-super@konakart.com*) is created for administering the more-technical configuration settings of the new store. Note that this new user will have broad Super-User privileges by default. By default, this user will be able to log on and administer all of stores in the mall by virtue of being a "Super User". Some customers may decide not to disclose the credentials of this newly-created "Super User" account, whereas others may decide to change the role that is assigned to this "Super User". You can define the role to be assigned in the Multi-Store Configuration - see details on configuring Multi-Store.

Chapter 7. Installation of KonaKart on other Application Servers

By default, KonaKart is provided with Tomcat, but it is actually designed to be usable on any compliant J2EE Application Server. This section provides some notes to help you install and configure KonaKart on a selection of other leading Application Servers.

Most Application Servers give users a variety of choices for application deployment. Some like to import EARs, others prefer WAR files. Some Application Servers accept "exploded" directory structures as the source for deployment whereas others require "un-explored" EAR or WAR files.

There is nothing in KonaKart that should prevent you from choosing the deployment system you prefer for your chosen Application Server.

In most cases the best way to begin is to run a complete installation of KonaKart from the GUI installer (or silent mode if you prefer), verifying and populating the database as part of this process. It can save you some time if you select the port number the target Application Server will use as the KonaKart port number during this installation process so that there is less configuration to do later on (for example, if your target Application Server is BEA WebLogic, you would choose to install KonaKart at port 7001).

Once KonaKart is installed you have the choice to use the expanded webapps for deployment or use the included ant build files to produce WAR and EAR files as you require.

In most cases, the easiest approach will be to use the ant scripts to produce an EAR file containing the 3 KonaKart webapps (*konakart*, *konakartadmin* and *birtviewer*). It can be advantageous to produce an EAR file as the whole application can be deployed in one go, rather than 3 separate WAR installations (which is, of course, perfectly possible, and may even be more-appropriate in some situations - see JBoss notes below).

To produce the EAR, (and the 3 WARs as a by-product), run the "*ant make_ear*" command in the "*custom*" directory of your KonaKart installation as in the example below:

For *WebLogic* add the "-Dweblogic=true" argument so that it builds a slightly-modified birtviewer War file that is compatible with WebLogic.

```
C:\Program Files\KonaKart\custom>.\bin\ant make_ear
Buildfile: build.xml

clean_wars:
    [echo] Cleanup WARS...
    [echo] Cleanup EARs...

make_wars:
    [mkdir] Created dir: C:\Program Files\KonaKart\custom\war
    [echo] Create konakart.war
    [war] Building war: C:\Program Files\KonaKart\custom\war\konakart.war
    [echo] Create konakartadmin.war
    [war] Building war: C:\Program Files\KonaKart\custom\war\konakartadmin.war
    [echo] Create birtviewer.war
    [war] Building war: C:\Program Files\KonaKart\custom\war\birtviewer.war

make_ear:
    [mkdir] Created dir: C:\Program Files\KonaKart\custom\ear
    [echo] Create konakart.ear
    [ear] Building ear: C:\Program Files\KonaKart\custom\ear\konakart.ear
    [echo] Create exploded version of konakart.ear
    [mkdir] Created dir: C:\Program Files\KonaKart\custom\konakart.ear
    [unzip] Expanding: C:\Program Files\KonaKart\custom\ear\konakart.ear into
                  C:\Program Files\KonaKart\custom\konakart.ear
    [move] Moving 1 file to C:\Program Files\KonaKart\custom\konakart.ear
    [move] Moving 1 file to C:\Program Files\KonaKart\custom\konakart.ear
    [move] Moving 1 file to C:\Program Files\KonaKart\custom\konakart.ear
    [unzip] Expanding: C:\Program Files\KonaKart\custom\konakart.ear\birtviewer.war.tmp into
                  C:\Program Files\KonaKart\custom\konakart.ear\birtviewer.war
    [unzip] Expanding: C:\Program Files\KonaKart\custom\konakart.ear\konakart.war.tmp into
                  C:\Program Files\KonaKart\custom\konakart.ear\konakart.war
    [unzip] Expanding: C:\Program Files\KonaKart\custom\konakart.ear\konakartadmin.war.tmp into
                  C:\Program Files\KonaKart\custom\konakart.ear\konakartadmin.war

BUILD SUCCESSFUL
Total time: 1 minute 0 seconds
```

In the following sections you will find guidance notes for installing KonaKart on selected Application Servers.

Note that from KonaKart v5.8.0.0 a java 1.6 (or later) runtime environment is required. Therefore KonaKart v5.8.0.0 will not run on App Servers which cannot support java 1.6 (or later) applications.

General Notes on Installing KonaKart on Application Servers

These general notes apply to installations on most application servers. Note any specific exceptions below in the sections dedicated to each Application Server.

Edit Config Files - Admin Application Functionality

In the KonaKart Admin Application there is a feature that allows you to define a set of files to view and edit from the Admin App. The files that are made available to view and edit are defined in the XML file called *konakart_config_files.xml*. This control file must be available on the classpath of the KonaKart Admin Application. In the default installation that includes Tomcat, this file is set up with relative paths to various properties files. This is hardly ever appropriate for other application servers so if this feature is required, the control file must be updated to suite your environment.

Rather than use relative pathnames for these files, it is often easier to use full path names.

Some examples of file references are: (note that on Windows you must use forward rather than backward slashes)

Installation of KonaKart on other Application Servers

WebSphere:

```
<file>
<displayName>KonaKart Properties File</displayName>
<fileName>
C:/Program Files/IBM/WebSphere/AppServer/profiles/AppSrv01/
installedApps/swindonNode01Cell/KonaKart.ear/konakart.war/WEB-INF/
classes/konakart.properties
</fileName>
</file>

(file name shown split over 3 lines but leave it on one line in the file)
```

WebSphere Community Edition:

```
<file>
<displayName>KonaKart Properties File</displayName>
<fileName>
C:/Program Files/IBM/WebSphere/AppServerCommunityEdition/repository/default/
Application_KonaKart/1216389520400/Application_KonaKart-1216389520400.car/
konakart.war/WEB-INF/classes/konakart.properties
</fileName>
</file>

(file name shown split over multiple lines but leave it on one line in the file)
```

JBoss: (for exploded deployments only)

```
<file>
<displayName>KonaKart Properties File</displayName>
<fileName>
C:/jboss-4.2.2.GA/server/default/deploy/konakart.war/WEB-INF/
classes/konakart.properties
</fileName>
</file>

(file name shown split over 2 lines but leave it on one line in the file)
```

WebLogic: (for exploded deployments only)

```
<file>
<displayName>KonaKart Properties File</displayName>
<fileName>
C:/Program Files/KonaKart/custom/konakart.ear/konakart.war/
WEB-INF/classes/konakart.properties
</fileName>
</file>

(file name shown split over 2 lines but leave it on one line in the file)
```

Glassfish:

```
<file>
<displayName>KonaKart Properties File</displayName>
<fileName>
C:/glassfish/domains/domain1/applications/j2ee-apps/konakart/
    konakart_war/WEB-INF/classes/konakart.properties
</fileName>
</file>

(file name shown split over 2 lines but leave it on one line in the file)
```

Email Properties File

If you use additional email properties (most people will not need to) you place these in a properties file called *konakart_mail.properties*. You have to define this filename in the KonaKart Admin Application under the *Configuration > Email Options* panel. You have to specify the full path name here (for the *KonaKart mail properties filename* field).

Some examples of mail properties file references are: (note that on Windows you must use forward rather than backward slashes)

Tomcat Default:

C:/Program Files/KonaKart/conf/konakart_mail.properties

JBoss:

C:/jboss-4.2.2.GA/server/default/conf/konakart_mail.properties

Reporting Port Numbers and Report Location

1. Set the port number for reporting as required (set these using the KonaKart Admin App under the *Configuration > Reports* section). Use the standard HTTP port for your application server, eg. 8080 (Tomcat/JBoss), 7001 (WebLogic), 9080 (WebSphere) etc..
2. Set the location (the "*Report definitions base path*") of the reports to your chosen location. This can be anywhere on the disk that's accessible to your application server. This could be the location where you first installed KonaKart in order to produce the EAR files. If so, you can leave the default for this field.

Configuring Parameters for Images

1. Set the port number for the image base URL as required (set these using the KonaKart Admin App under the *Configuration > Images* section). Use the standard HTTP port for your application server, eg. 8080 (Tomcat/JBoss), 7001 (WebLogic), 9080 (WebSphere) etc..
2. In a similar fashion to the above, set the image base path to the appropriate location for your environment. Typically, this will be the images directory under your konakart webapp. Some examples:

Tomcat Default:

C:/Program Files/KonaKart/conf/konakart_mail.properties

JBoss:

C:/jboss-4.2.2.GA/server/default/deploy/konakart.war/images

Setting the Optimum Memory Values

If you find that you run out of memory you will need to define more memory for your Application Server. This can be done in a variety of ways, so check the documentation for your chosen Application Server for setting memory flags.

When setting memory settings by defining values for CATALINA_OPTS or JAVA_OPTS you might need to set values such as these:

-XX:PermSize=256m -XX:MaxPermSize=256m -Xms512m -Xmx1024m

(You may find you need different settings for your particular environment).

Some Application Servers allow you to define JVM parameters from within their Administrative Consoles. An example of this is WebSphere where you need to navigate to *Server > Server Infrastructure > Java & Process Management > Process Defn* then set your values for Maximum Heap Size (eg. 1024) and Initial Heap Size (eg. 512). (You may need require different settings to suit your own environment).

Installing KonaKart on BEA's WebLogic Application Server

Verified on WebLogic 10.0 MP1 on Windows Vista and Windows 7

Verified on WebLogic 10.3.5.0 (11g) on Windows 7 (with exploded EAR files)

Refer to the general notes for installing KonaKart on all Application Servers .

Installation

Use the deployment method of choice. If you want to be able to edit KonaKart configuration files (eg konakart.properties) without having to re-deploy the application you will have to install using exploded directories. If you don't need this functionality, you can simply deploy KonaKart as an EAR file.

When using the custom ANT command to produce the EAR for WebLogic add the *-Dweblogic=true* argument so that it creates a compatible birtviewer.war for WebLogic.

Deploying using WebLogic's *Install Application Assistant* in the Administration Console is particularly straightforward but WebLogic has a variety of deployment options you could also use (such as the autodeploy facility or their command-line driven deploy tool).

Follow these steps when using the Administration Console for deployment: (as applicable for your version of WebLogic)

"Lock and Edit" then choose "Install", then browse to your EAR file (for non-exploded deployments) or to the directory (a directory called konakart.ear under the custom directory of your KonaKart installation) where your exploded version of the EAR is located (for exploded deployments).

If you used the ant build file to create the EAR file and the exploded EAR file these will be located under your KonaKart installation directory at:

1. /custom/ear/konakart.ear (the non-exploded EAR file)
2. /custom/konakart.ear (the exploded EAR file directory)

Ensure that your EAR is on a directory path that does not contain spaces (such as with "Program Files" on Windows).

Once selected, proceed through the wizard. For "Targeting style" choose "Install this deployment as an application". Accept all other defaults, click on "Finish", then finally click on "Activate Changes" when successful.

Eventually, you should see the message "*All changes have been activated. No restarts are necessary.*" and there should be no errors in the WebLogic log.

Next, click on the checkbox next to konakart and start the application by choosing "Start Servicing All Requests" from the "Start" dropdown list. Confirm this action on the next screen and check the WebLogic log for any errors.

Configuration

1. Set the port numbers (7001) for the reporting configuration variables as required (set these using the KonaKart Admin App under the *Configuration > Reports* section)
2. To set-up KonaKart to use SSL you need to set it up in the KonaKart Admin App and the BEA WebLogic Administration Console as follows:

Set the HTTP (7001) and HTTPS (7002) port numbers as required (set these using the KonaKart Admin App under the *Configuration > HTTP/HTTPS* section)

To enable SSL in WebLogic first click the 'Lock and Edit' button under the Change Center. Under the "Domain Structure" navigate through "Environment" then "Servers" then click on your server listed under the "Servers" table on the right hand side. (The default server will be called "AdminServer").

On the General Configurations panel, check the "SSL Listen Port Enabled" checkbox. Leave the port number for SSL as 7002 and ensure this matches the port number specified above in the KonaKart Admin Application.

Click on the "Save" button, then click on "Activate Changes" to save these changes in your WebLogic environment.

3. Set the memory parameters to suit your environment. Set your memory arguments inside the setDomainEnv.cmd (Windows) or setDomainEnv.sh (Unix/Linux) files.

Set a new USER_MEM_ARGS environment variable with your chosen memory flags, or modify the existing MEM_FLAGS environment variable assignment to suit your requirements. Take note of the code that follows the assignment to MEM_FLAGS - you might find that then your values are overridden.

Restart WebLogic for the changes to take effect and double-check the beginning of the WebLogic log to see what memory settings you actually achieved!

Installing KonaKart on JBoss

Verified on JBoss 4.2.2 GA through JBoss 7.1.1

Refer to the general notes for installing KonaKart on all Application Servers .

Installation

For JBoss we recommend that you deploy KonaKart as an exploded EAR file, or exploded WAR files so that you can guarantee the full path names of the various configuration files. This is useful so that you can

define these full path names and subsequently edit these files from within the comfort of the KonaKart Admin Application.

Use the ANT targets: "make_ear -Djboss=true" to produce your EAR for JBoss.

1. The KonaKart applications can be deployed in the 'server configuration' directory of your choice. In this example we will use the 'default' configuration. In this example we also assume a Windows installation in the *C:\jboss-4.2.2.GA* directory.
2. Here we follow the procedure for the 3 WAR installation (but for the exploded EAR procedure follow an almost identical path). Create three new directories under *C:\jboss-4.2.2.GA\server\default\deploy* :
 1. konakart.war
 2. konakartadmin.war
 3. birtviewer.war
3. Expand konakart.war into the konakart.war directory and do the same for konakartadmin and birtviewer.
4. Start JBoss.
5. Check the JBoss log to ensure that there are no errors.

If you do not deploy as 3 separate exploded WAR files, JBoss generates its own temporary directories to hold these deployments. This wouldn't be so bad except that JBoss creates *new* temporary directories every time you restart JBoss.

It's still possible to deploy KonaKart into JBoss using an "unexploded" konakart.ear file but you will not be able to use the "Edit Config Files" functionality in the KonaKart Admin Application. If you wish to deploy using the EAR file you simply drop it into the deploy directory of your JBoss installation for automatic deployment. Check your JBoss server log; there should be no errors during this deployment.

Configuration

1. Set the port numbers (8080) for reporting as required (set these using the KonaKart Admin App under the *Configuration > Reports* section)
2. Set the HTTP (8080) and HTTPS (8443) port numbers as required (set these using the KonaKart Admin App under the *Configuration > HTTP/HTTPS* section). You will also have to setup SSL in JBoss if you haven't already done so (refer to JBoss documentation to find out how to do this). A quick way to do this if you don't have a keystore already (and for development purposes only) is to copy the *conf/.keystore* file from your KonaKart installation to the *C:\jboss-4.2.2.GA\server\default\conf* directory of JBoss, then edit your *C:\jboss-4.2.2.GA\server\default\deploy\jboss-web.deployer\server.xml* file as follows:

```
<!-- Define a SSL HTTP/1.1 Connector on port 8443
This connector uses the JSSE configuration, when using APR, the
connector should be using the OpenSSL style configuration
described in the APR documentation -->

<Connector port="8443"
    protocol="HTTP/1.1"
    SSLEnabled="true"
    clientAuth="false"
    sslProtocol="TLS"
    keystoreFile="conf/.keystore"
    keystorePass="kkpassword" />
```

3. Set the memory parameters to suit your environment (eg. I set `JAVA_OPTS=-XX:PermSize=256m -XX:MaxPermSize=256m -Xms512m -Xmx1024m`).

Installing KonaKart on IBM's WebSphere Application Server

Verified on WebSphere 6.1 on Windows Vista

Refer to the general notes for installing KonaKart on all Application Servers .

Usage Limitation: Reports with embedded charts produced using EMF 2.2 cannot be run in the default WebSphere 6.1 installation due to a conflict between versions of EMF (WebSphere 6.1 uses 2.1 whereas KonaKart/BIRT use EMF 2.2). If reports with embedded charts are important to you, you can either write new, compatible, reports using EMF 2.1 or modify the class-loading mode of the KonaKart application to parent last (as described in various reports on the subject on the Internet).

Note that the default download kit of KonaKart contains code compiled with a java 1.6 target therefore it will not work on WebSphere 6.0 which uses a 1.4 JRE.

Installation

Use the deployment method of choice. Loading using an EAR is probably the simplest by navigating to Enterprise Applications in the Administrative Console, "Install" application, browse to your EAR file, then proceed through the wizard accepting all the defaults.

Configuration

1. If you are not using a Sun JRE for WebSphere, you will need to add two jars `jsse.jar` and `jce.jar` from your Sun JRE to support some SSL and mail functionality of KonaKart. Either package these in the WARs of `konakart.war` and `konakartadmin.war`, or place them in the lib directories of these applications once deployed.
2. Set the port numbers (9080) for the reporting configuration variables as required (set these using the KonaKart Admin App under the *Configuration > Reports* section)
3. Set the HTTP (9080) and HTTPS (9443) port numbers as required (set these using the KonaKart Admin App under the *Configuration > HTTP/HTTPS* section)
4. Set the memory parameters to suit your environment. Use the WebSpehere Administrative Console to set the JVM parameters. Navigate to *Server > Server Infrastructure > Java & Process Management > Process Defn* then your values for Maximum Heap Size (eg. 1024) and Initial Heap Size (eg. 512). (You may need require different settings to suit your own environment).

Installing KonaKart on IBM's WebSphere Application Server Community Edition

Verified on WebSphere ASCE 2.0.0.2 on Windows Vista

Refer to the general notes for installing KonaKart on all Application Servers .

Installation

Load the KonaKart EAR using the Administrative console, accepting all defaults. (Some XML descriptor validation warnings appear in the console log but these can be ignored - there are no *geronimo-web.xml* files included).

Configuration

1. Set the port numbers (8080) for reporting as required (set these using the KonaKart Admin App under the *Configuration > Reports* section)
2. Set the HTTP (8080) and HTTPS (8443) port numbers as required (set these using the KonaKart Admin App under the *Configuration > HTTP/HTTPS* section)
3. Set the memory parameters to suit your environment (eg. I set *JAVA_OPTS=-XX:PermSize=256m -XX:MaxPermSize=256m -Xms400m -Xmx700m*)

Installing KonaKart on GlassFish

Verified on GlassFish v2 UR2 b04 (also known as Sun Java System Application Server 9.1_02 b04-fcs), Glassfish 3.1.2 and Glassfish 4.0.

Refer to the general notes for installing KonaKart on all Application Servers .

Installation

Assuming a vanilla installation of Glassfish, simply place the KonaKart EAR in the *domains/domain1/autodeploy/* directory of the GlassFish installation. The KonaKart EAR is loaded automatically by GlassFish. Check the GlassFish server log to ensure there are no errors.

Alternatively you can deploy the EAR file by using the GlassFish admin console (typically at <http://localhost:4848>)

Configuration

1. Set the port numbers (8080) for reporting as required (set these using the KonaKart Admin App under the *Configuration > Reports* section)
2. Set the HTTP (8080) and HTTPS (8181) port numbers as required (set these using the KonaKart Admin App under the *Configuration > HTTP/HTTPS* section). You can use the default SSL setup of GlassFish for development purposes.
3. If you find you run out of memory, adjust the JVM memory parameters to suit your environment. The easiest way to do this with GlassFish is by using the Admin Console under *Configurations > server-config > JVM Settings*
4. Configure the image locations in the Admin App under *Configuration > Images*. Set the image base url to <http://localhost:8080/konakart/images/> and the image base path to something like *C:/glassfish/domains/domain1/applications/j2ee-apps/konakart/konakart_war/images* (or the equivalent in your installation).

Installing KonaKart on JOnAS with Tomcat

Verified on JOnAS v4.9.2 / Apache Tomcat v5.5.26 on Windows Vista

Refer to the general notes for installing KonaKart on all Application Servers .

Installation

Place the KonaKart EAR in the *apps/autoload* directory of the JOnAS installation. The KonaKart EAR is loaded automatically by JOnAS (some warnings about the use of DTDs can be ignored: "konakart.ear is using DTDs, WsGen needs Schema only : WEB-INF/web.xml use a DTD").

Configuration

1. Set the port numbers (9000) for reporting as required (set these using the KonaKart Admin App under the *Configuration > Reports* section)
2. Set the HTTP (9000) and HTTPS (9043)port numbers as required (set these using the KonaKart Admin App under the *Configuration > HTTP/HTTPS* section). You will also have to setup SSL in JOnAS/Tomcat if you haven't already done so (refer to JOnAS and Tomcat documentation to find out how to do this). A quick way to do this if you don't have a keystore already (and for development purposes only) is to copy the *conf/.keystore* file from your KonaKart installation to the *conf* directory of JOnAS, then add to your server.xml file as follows:

```
<!-- Define a SSL Coyote HTTP/1.1 Connector on port 9043 -->  
  
<Connector port="9043"  
           maxHttpHeaderSize="8192"  
           maxThreads="150"  
           minSpareThreads="25"  
           maxSpareThreads="75"  
           enableLookups="false"  
           disableUploadTimeout="true"  
           acceptCount="100"  
           scheme="https" secure="true"  
           clientAuth="false"  
           sslProtocol="TLS"  
           keystoreFile="conf/.keystore"  
           keystorePass="kkpassword" />
```

3. Set the memory parameters to suit your environment (eg. I set *JONAS_OPTS=-XX:PermSize=256m -XX:MaxPermSize=256m -Xms400m -Xmx700m*). ("Your mileage may vary").
4. Configure the image locations in the Admin App under *Configuration > Images* . Set the image base url to <http://localhost:9000/konakart/images/> and the image base path to something like *C:/JOnAS-4.9.2/tomcat/work/webapps/jonas/ear/konakart/konakart/images* (or the equivalent in your installation).

Installing KonaKart on JOnAS with Jetty

Verified on JOnAS v4.9.2 / Jetty v5.1.10 on Windows Vista

Refer to the general notes for installing KonaKart on all Application Servers .

Installation

Place the KonaKart EAR in the *apps/autoload* directory of the JOnAS installation. The KonaKart EAR is loaded automatically by JOnAS (some warnings about the use of DTDs can be ignored: "konakart.ear is using DTDs, WsGen needs Schema only : WEB-INF/web.xml use a DTD").

Configuration

1. Set the port numbers (9000) for reporting as required (set these using the KonaKart Admin App under the *Configuration > Reports* section)
 2. Set the HTTP (9000) and HTTPS (9043)port numbers as required (set these using the KonaKart Admin App under the *Configuration > HTTP/HTTPS* section). You will also have to setup SSL in JOnAS/Jetty if you haven't already done so (refer to JOnAS and Jetty documentation to find out how to do this). A quick way to do this if you don't have a keystore already (and for development purposes only) is to copy the *conf/.keystore* file from your KonaKart installation to the *conf* directory of JOnAS, then add to edit your *jetty5.xml* file as follows:

- Set the memory parameters to suit your environment (eg. I set `JONAS_OPTS=-XX:PermSize=256m -XX:MaxPermSize=256m -Xms400m -Xmx700m`). ("Your mileage may vary").
 - Configure the image locations in the Admin App under *Configuration > Images* . Set the image base url to `http://localhost:9000/konakart/images/` and the image base path to something like `C:/JOnAS-4.9.2/jetty/work/webapps/jonas/ear/konakart/konakart/images` (or the equivalent in your installation).

Chapter 8. Upgrading

KonaKart has been designed with upgrading in mind. If you customize KonaKart using the recommended techniques (particularly if you've used the APIs) the upgrade process should be a fairly straightforward process. This section provides advice on the processes to follow in order to achieve a painless upgrade.

Of fundamental importance is that the KonaKart APIs remain backwards-compatible between releases. (Very occasionally a "backwards-compatibility warning" is issued when an API changes but this is very rare).

This means that code that you have written against the APIs will continue to work for a later version. Therefore, your investment in the code you have written is preserved.

This is in contrast to many other eCommerce systems where customisations are deeply-integrated with the core system resulting in extremely costly and time-consuming upgrades.

You may have written your storefront application in a certain technology of your choice (for example Apache Wicket, JSF etc) that uses the KonaKart APIs. In theory, the upgrade process will simply be a matter of replacing the jars from the earlier version with the new jars in the version you wish to upgrade to. Your storefront may not be able to take advantage of the new features provided in the new version (without modifying your storefront application to use the new features) but your storefront should continue to work as before because the APIs are backwards-compatible. In practice the upgrade process can sometimes be a little more complicated than just replacing the jars and some of the more complicated cases are discussed below.

Recommended Upgrade Steps

It's difficult to define an ideal set of steps for every situation as the approach you take depends a lot on what your starting position is. The following should be taken as a set of recommended steps to consider in the overall process but not all will be applicable to every upgrade.

Before starting it is advised that you read the release notes for the version you are planning to upgrade to. These can be found at <http://www.konakart.com/downloads/release-notes> [<http://www.konakart.com/downloads/release-notes>]

Backup

It is always recommended that you backup all data and code belonging to the version that you wish to upgrade. Ensure you have backups before proceeding.

Installation of the new Version

Install the new version alongside the old version (use a new database schema so that they don't collide). Installation using the graphical wizards is recommended for one-off installations but the silent installation may be preferable in highly-automated environments where installations need to be repeated several times.

This new installation will include the default version of the new storefront client and any new directories, configuration files, documentation and jars provided in the new version.

This new installation will provide an extremely useful reference platform that you can use for a variety of purposes such as verifying the behaviour of new features.

All the updated source code provided with KonaKart will be included in this new installation so if any of these items have been customised these customisations may need to be merged with the new versions.

Database Upgrade

An upgrade SQL file (one for each database type) is provided that needs to be run to upgrade the database from one version to the next. If you are jumping a few versions you must run each upgrade SQL file in sequence in order to prepare the database for the new version. The database upgrade scripts will not destroy your data (customers, orders, products etc) but it is highly-recommended to run the upgrade process on a test environment before applying it to any production database.

On Windows, (assuming KonaKart was installed in the default location), the database upgrade scripts for MySQL can be found in the following directory:

C:/Program Files/KonaKart/database/MySQL

Test new Installation against Upgraded Database

Having upgraded the database from the previous version you can verify that the new version of KonaKart runs fine against the upgraded database by changing the database parameters in the konakart.properties and konakartadmin.properties files in the new installation's webapps (konakart and konakartadmin webapps).

Re-build Custom Code

If you have custom code (including your own order total, shipping and payment modules, one-page-checkout code etc) that uses the KonaKart libraries you will need to recompile it against the new jars.

Exactly how you go about this task will depend on various factors including how you have structured your development environment and build system.

If you use the supplied ANT build files to build your custom code you should copy your custom sources across to the new custom directory and build from there. This will pick up the new ANT build file and the new libraries that will be required. Depending on what you have done in your custom code it is possible that you may have compilation errors to fix at this stage.

If you have your own build system in place for your custom code you should copy the dependent jars from the new installation to your custom build system and rebuild. Again, depending on what you have done in your custom code, it is possible that you may have compilation errors to fix at this stage.

Upgrade the Struts Storefront

Of all the upgrade tasks this can be the most complex as it involves the careful comparison of your storefront code with the storefront code in the new version.

Because of the backwards compatibility of the APIs in theory you should not need to change your modified storefront at all and it will continue working as before.

You only need to change your storefront code if you want to upgrade your storefront to the new storefront code which will use the new features introduced in the new version of KonaKart.

If you wish to upgrade your storefront you will have to merge your own changes with the new code in the following areas:

- Struts Configuration Files

- Struts Action Classes, business logic and forms
- JSPs

The best way to execute the merges will depend on how many changes you've made. If you have made a very large number of changes in these areas it's likely that the easiest option is to merge the new storefront features into your current storefront code. If, on the other hand, you have made very few changes to the storefront code, the best option is probably to merge those few changes into a copy of the new storefront code.

Assess Changes to WebApp Configuration files

Sometimes changes are made to the webapp configuration files. An example is when JSON was introduced a new section was introduced into the web.xml file for the konakart webapp to define the new servlet.

These key configuration files should be compared with your current versions to assess whether the changes are required. Files to pay particular attention to are:

- \${KONAKART_HOME}/webapps/konakart/WEB-INF/web.xml
- \${KONAKART_HOME}/webapps/konakartadmin/WEB-INF/web.xml
- \${KONAKART_HOME}/webapps/birtviewer/WEB-INF/web.xml

Assess Changes to Tomcat Configuration files

Some releases include an upgrade of tomcat. If you use tomcat for KonaCart you will probably want to upgrade to the same version of tomcat and move your current system across to the new tomcat installation.

Some releases use the same version of tomcat (as the previous version) but have modified configuration files that should be compared with your current versions to assess whether the changes are required. Files to pay particular attention to are:

- \${KONAKART_HOME}/conf/context.xml
- \${KONAKART_HOME}/conf/server.xml
- \${KONAKART_HOME}/conf/web.xml

KonaCart Directory Structure Changes

Sometimes new releases provide new functionality that requires a new directory structure. An example is when Solr was first introduced a new solr directory was added at the base of the KonaCart home directory. Check your reference installation for any new directories and consider introducing them into your existing structure (where you place them will depend a lot on which App Server you happen to be using).

KonaCart Message Changes

Most new releases introduce new message strings that may need to be translated into the languages you need to support in your system. The message files to consider are:

- \${KONAKART_HOME}/webapps/konakart/WEB-INF/classes/Messages.properties
- \${KONAKART_HOME}/webapps/konakartadmin/WEB-INF/classes/AdminMessages.properties

- \${KONAKART_HOME}/webapps/konakartadmin/WEB-INF/classes/AdminHelpMessages.properties

To help identify the new messages introduced between particular versions of KonaKart you can refer to the files in the \${KONAKART_HOME}/utils/kkMessages/new-Messages , \${KONAKART_HOME}/utils/kkMessages/new-AdminMessages and \${KONAKART_HOME}/utils/kkMessages/new-AdminHelpMessages directories. When new messages are introduced between the specified versions in the above 3 Message properties file types a file is created that contains only the newly introduced messages.

If a particular **Message_X.X.X_to_Y.Y.Y.Y.properties* properties file is missing from the \${KONAKART_HOME}/utils/kkMessages/new-* directory it means that no new messages of that type were introduced between the two versions.

Documentation and javadoc Changes

It is highly-recommended that the "doc" directory of the new installation is checked for the latest documentation. The User Guide will be updated for every release and additional documentation may be added to this directory covering specific areas. Note also that the javadoc will change in each new release so if you refer to a local copy for the javadoc you should reference that in the new installation under *webapps/javadoc/* .

Admin App Changes

New panels are often added to the Admin App in new versions of KonaKart. You need to check that the roles that you have defined for your Admin App users are appropriate for your needs. You may find that you need to allow access to some or all of the new panels otherwise they will not become visible to your Admin App users. It can be useful to install a fresh copy of KonaKart alongside your development system (use a new database schema) to allow you to refer to the default set-up for the Admin App. This default set up will show you all the roles that are assigned to the *admin@konakart.com* user with a standard default installation.

Chapter 9. Administration and Configuration

This chapter seeks to explain the many different ways in which KonaKart can be configured.

Most of the Administration and Configuration of KonaKart can be carried out using the KonaKart Administration Application.

KonaKart Administration Application

KonaKart includes a sophisticated browser based administration application. It uses AJAX technology to provide a snappy user interface while maintaining the advantages of running the application from a browser. Each application window has an on-line help facility which is the first place to look in order to understand the available functionality.

It incorporates a security subsystem with role based security. Each user can be assigned one or more roles that determine access to the available functionality with read / insert / edit and delete granularity. The user name / password based access, has the facility to block users for a programmable period after a number of unsuccessful login attempts.

Auditing may be enabled for all Admin App API calls with two levels of detail. All audit data is stored in the KonaKart database and may be browsed and filtered through the Admin App.

The admin application is fully internationalized and can be translated via a message catalog. Each panel has an online help facility that explains the functionality available.

KonaKart
Enterprise Java eCommerce

Welcome back, admin@konakart.com
[Set Language](#) [Change Password](#) [Sign Out](#) [About](#)

My Store Status	
My Store Status	My Store Status
Latest News	
Configuration	
Products	
Modules	
Customers	
Orders	
Marketing	
Locations/Taxes	
Localizations	
Tools	
Reports	
Audit Data	
Custom	

My Store Status	
Delivered Orders	0
Partially Delivered Orders	0
Payment Declined Orders	0
Payment Received Orders	1
Pending Orders	1524
Processing Orders	0
Waiting for Payment Orders	0
Number of Customers	1361
Number of Products	27
Number of Manufacturers	9

Orders in Last 30 Days

Date	Orders
7-Dec	3
8-Dec	1
9-Dec	2
10-Dec	3
11-Dec	2
12-Dec	2
13-Dec	1
14-Dec	2
15-Dec	3
16-Dec	3
17-Dec	4
18-Dec	2
19-Dec	5
20-Dec	2
21-Dec	0
22-Dec	0
23-Dec	0
24-Dec	0
25-Dec	1
26-Dec	1
27-Dec	0
28-Dec	3
29-Dec	3
30-Dec	4
31-Dec	4
1-Jan	1
2-Jan	0
3-Jan	3

[Refresh](#)

KonaKart Admin Application - Status View

Main Features

The main features of the admin app are:

- Store status summary (i.e. number of orders, number of products etc.)
- Store maintenance
 - Create new stores
 - Edit existing stores
- Change state of stores (i.e. enable / disable, maintenance mode)
- Delete stores

- Maintenance of configuration variables
- Product maintenance
 - Product Category maintenance
 - Product Option maintenance
 - Product Manufacturer maintenance
 - Product Tag Group and Tag maintenance
 - Product Payment Schedule maintenance
 - Product Review maintenance
 - Product Custom Attribute Template and Custom Attribute maintenance
 - Product Catalog maintenance
 - Miscellaneous Item Type and Item maintenance
- Installation and removal of modules (payment, shipping, order total and other modules)
- Customer maintenance
 - Send email
 - Role maintenance
 - Reset Password
 - Login to eCommerce application on behalf of a customer. Useful for call center applications.
 - Customer Group maintenance
 - Customer review maintenance
 - Customer booking maintenance
- Orders
 - Generate invoice (template based)
 - Generate packing slip (template based)
 - Change state of order and send email
 - View all payment gateway notifications associated with order
 - Manage returns
- Marketing
 - Promotion maintenance
 - Coupon maintenance
 - Customer Tag and Expression maintenance

- Customer Communications where you can send template based eMails to all customers, to all customers who have requested to receive the newsletter, to customers that have asked to be notified about any updates for a particular product and to customers belonging to a particular group or satisfying a certain expression.
- Locations / Taxes
 - Country maintenance>
 - Zone maintenance
 - Tax Area maintenance
 - Tax Class maintenance
 - Tax Rate maintenance
- Localizations
 - Currency maintenance
 - Language maintenance
 - Message maintenance
 - Order Status maintenance
 - Address Format maintenance
- Reports
- View Audit Data
- Tools
 - Delete expired sessions
 - Refresh caches
 - Manage Solr search engine
- Custom panels - Add custom panels that implement your custom business logic.

Reporting

The KonaKart admin application provides powerful reporting functionality through integration with BIRT [<http://www.eclipse.org/birt/phoenix/>] , the very popular open source Business Intelligence and Reporting Tool. Although an ever expanding list of useful reports is provided in the KonaKart download, the integration is done in such a way that allows users and system integrators to develop and customize their own reports by using the BIRT Eclipse based development environment.

Reporting - BIRT Viewer Security

By default the BIRT Viewer webapp is protected by a configurable layer of security that ensures that only suitably-authorised Administrators with active sessions are permitted to execute the reports using the BIRT Viewer webapp.

To disable the security completely you can set the "securityEnabled" initialisation parameter to "false" in the birtviewer web.xml.

When security is enabled, a user is allowed access to the reports through BIRT viewer if and only if:

- A valid sessionId is passed as a parameter to the birtviewer webapp. A valid sessionId is one that exists in the database for the specified user and store and that it hasn't expired.
- The user has been granted access to run the reports through role-based security.

To be able to run the reports the user must be assigned a role that permits that user to execute the reports. This is defined on the privileges screen of the role-based security section of the Admin Application (Under Customers >> Maintain Roles).

The "custom1" flag must be unticked to allow the user to run the reports:

Refresh Config Cache	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Reports	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Reports Configuration	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	If set reports cannot be run

Reports Privileges

You can easily customise the pages that users are redirected to in the event of session expiry and unauthorised access. The URLs of the pages that are used are defined in the birtviewer webapp's web.xml file.

Role-based Security and Configuration

Many panels in the admin application may be configured to display or hide certain fields and buttons. The configuration is set by selecting a role in the Maintain Roles panel and then by clicking on the Privileges button on the same panel. A pop-up panel should appear similar to the image shown below:

Customer Communications	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Customer Details	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Customer For Order	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Customer Groups	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Customer Orders	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Customers	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Delete Expired Sessions	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Digital Downloads	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Edit a Store in a Mall	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Edit Customer	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Edit Order	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Edit Product	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Email Options	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Geo-Zones	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Role Privileges

Each panel has a number of checkboxes to assign privileges. The standard privileges are Insert, Edit and Delete, although some panels have custom privileges which are highlighted in green. In order to understand what a green highlighted checkbox refers to, a yellow popup will appear when you move your mouse over it. For example the Edit Order panel has a couple of configuration options which are:

- Enable the read and edit of credit card details.
- Enable the read and edit of custom fields, order number and tracking number.

File-based Configuration

It is possible to configure the Admin Application on a global basis by defining certain properties in the konakartadmin_gwt.properties file (which can be found in the classes directory of the konakartadmin webapp).

The configurations you make in this file-based technique are for every user of the Admin App (deployed in the associated webapp) no matter what roles are defined for each user.

By default, the file-based configuration ("FBC") properties are commented out and as such have no effect. To enable them you need to uncomment the relevant line(s).

The supported configuration properties are defined in the konakartadmin_gwt.properties file and will be updated over time. Here is a sub-set of the currently-supported configuration options:

```

# -----
# Enterprise Feature
# File-based Configuration
# These settings make global changes to the Admin App for all users

#fbc.kk_panel_communications.hide_expression_selection = true

# Use this to set the default for the "Use Customer Language" checkbox
# (default is true if not defined)
#fbc.kk_panel_communications.default_use_cust_lang = false

#fbc.kk_panel_editProduct.hide_attributes_tab = true

#fbc.kk_panel_products.hide_name_show_sku = true

#fbc.kk_panel_editCustomer.address.hide_city = true
#fbc.kk_panel_editCustomer.custom.hide_custom1 = true

#fbc.kk_panel_editCustomer.personal.hide_customerGroup = true
#fbc.kk_panel_editCustomer.personal.hide_dateOfBirth = true
#fbc.kk_panel_editCustomer.personal.hide_fax = true
#fbc.kk_panel_editCustomer.personal.hide_first_name = true
#fbc.kk_panel_editCustomer.personal.hide_gender = true
#fbc.kk_panel_editCustomer.personal.hide_last_name = true
#fbc.kk_panel_editCustomer.personal.hide_newsletter = true
#fbc.kk_panel_editCustomer.personal.hide_state = true
#fbc.kk_panel_editCustomer.personal.hide_tel = true
#fbc.kk_panel_editCustomer.personal.hide_tel_other = true
#fbc.kk_panel_editCustomer.personal.hide_type = true
#fbc.kk_panel_editCustomer.personal.hide_visibility = true

#fbc.kk_panel_editCustomer.hide_address_tab = true
#fbc.kk_panel_editCustomer.hide_custom_tab = true
#fbc.kk_panel_editCustomer.hide_points_tab = true
#fbc.kk_panel_editCustomer.hide_tags_tab = true

#fbc.g.kk_panel_login.enter_store_as_text_not_droplist = true

#fbc.kk_panel_promRules.hide_categories = true

# Hide the Print button on the Order Invoice view
#fbc.kk_display_panel.invoice.hide_print_btn = true

# Stops logout after a browser refresh
#fbc.save_session_in_cookie = true

# etc...
# check your own kit to discover the properties available in your version

```

After uncommenting a property it is necessary to refresh the caches (you can do this from the Tools section of the Admin App) then refresh your browser so that the changes to the Admin App User interface will be enabled.

Product Image Uploads

It is possible to define the way images are created during the image upload process using File-based Configuration.

By default, every time a product image is imported, 4 images are created from the imported image scaled to 4 different sizes. The number of images created and the sizes of each can be defined in konakartadmin_gwt.properties. Other characteristics (such as maximum number of images to display per product and directory structure) can also be defined.

The supported configuration properties are defined in the konakartadmin_gwt.properties file and will be updated over time. Here are the currently-supported configuration options for image uploads:

```

# Image Scaling
# Only relevant to the images.tab.version = 2 (new images tab introduced with v6.5.0.0)
# Default, if not specified is "big;360;360 medium;150;150 small;80;80 tiny;60;60"
#
# For each size defn this is   name;height;width
#
# This means that for any uploaded image these four images are created with the following
# characteristics:
# Image 1: {product_UUID}_1_big.XXX (360x360 pixels)
#           : {product_UUID}_1_medium.XXX (150x150 pixels)
#           : {product_UUID}_1_small.XXX (80x80 pixels)
#           : {product_UUID}_1_tiny.XXX (60x60 pixels)
# XXX = the original file extension (used if add_extension is not set to false - see below)
#
# Retaining the original aspect ratio:
# Use big;;300      to create an image with width 300 pixels and unspecified height to
#                      retain aspect ratio
# Use big;200;     to create an image with height 200 pixels and unspecified width to
#                      retain aspect ratio
#
# Create a version identical to the original:
# Use big;;      to create an image with the same height and width as the original

# For best results order the images definitions from large to small
#fbc.kk_panel_editProduct.images.options = big;360;360 medium;150;150 small;80;80 tiny;60;60

# Defines whether or not to append a period and an extension to the generated image file
# names:
#fbc.kk_panel_editProduct.images.add_extension          = false

# Defines how many images are displayed for editing on the Edit Product panel (default is 8)
#fbc.kk_panel_editProduct.images.max                   = 8

# Defines the depth of the directory tree used for constructing image file names (default
# is 4)
# If 0 is used, all images will placed in the same directory under the Image Base Path
# If >0 the file path is created by using directories named by the first n characters of the
# UUID
# The purpose of the directory tree for images is to avoid having too many files in each
# directory so you should choose use a high value for the depth if you have a very large
# number of images.
#fbc.kk_panel_editProduct.images.dir.depth            = 4

# Defines the name of a directory that will be used to construct a filename for storing
# the product images. This directory (defaults to "prod") will be added to the Image Base
# defined for the store.
# It can be left blank if you want no product image directory added at all.
#fbc.kk_panel_editProduct.images.dir.name             = prod

```

It's better if you can decide what your image formats will be before you load all your images as the definitions only affect the scaling that takes place after product images are uploaded. Making a change to the image scaling and creation configuration parameters will not affect existing product images (but you can reload them if you wish).

Manufacturer Image Uploads

It is possible to define where manufacturer images are created during the image upload process using File-based Configuration.

By default, manufacturer images are uploaded to a filename that is a concatenation of the image base path (a configuration variable defined in the Administration Application) a "manufacturer" image directory name (defined in File-based Configuration and defaulting to "manufacturer") and the filename itself. The target directory and filename (except the image base path) is displayed in the image upload dialogue after selecting an image from the local file system. This directory and filename can be modified to change the target location that the file will be uploaded to.

The manufacturer directory configuration property is defined in the konakartadmin_gwt.properties file as follows:

```
# Defines the name of a directory that will be used to construct a filename for storing the
# manufacturer images. This directory (defaults to "manufacturer") will be added to the
# Image Base defined for the store. It can be left blank if you want no manufacturer image
# directory added at all.
#fbc.kk_panel_manufacturers.images.dir.name = manufacturer
```

The manufacturer directory configuration property is only used for creating new images. Once the image has been uploaded the file location is saved in the database in the manufacturer table.

Launching the Admin App

Normally, whenever the Admin App is launched using the standard URL (<http://localhost:8780/konakartadmin/>), the administrator is presented with a login panel where he must enter his credentials in order to gain access to the application. In a multi-store environment there is also a drop list of store names so that the administrator may choose the store that he wants to log in for.

By using a Launcher servlet, the startup process of the Admin App may be modified as follows:

The store id may be passed to the launcher servlet so that the administrator is not given the possibility to choose a store from the drop list. The drop list is no longer shown on the login panel. An example of calling the launcher for storeId = store1:

<http://localhost:8780/konakartadmin/launcher?storeId=store1>

In order to achieve Single Sign On, credential checking may be delegated to the AdminAppAuthenticationMgr class. This class has a method to authenticate the request and a method that returns a URL that may redirect to an SSO login screen if the administrator isn't logged in. If the authentication method determines that the customer is already logged in, then the Admin App will start without displaying the login screen at all. Note that the user identified by the userId returned by the authentication method, must exist in the KonaKart database as an Admin User.

The source code of the launcher and authentication manager are supplied in the /KonaKart/custom/adminnappn/src/com/konakartadmin/servlet directory.

Configuring KonaKart for HTTPS / SSL

SSL can be enabled and configured in the *Configuration>>HTTP/HTTPS* section of the Admin App. When SSL is enabled and the port numbers are correctly defined, KonaKart will automatically switch between HTTP and HTTPS depending on whether the customer is logged in or not. Whenever the customer is logged in, a session id is passed back and forth, so the protocol is set to HTTPS. Whenever the customer isn't logged in, the protocol is set to HTTP.

Editing the KonaKart Configuration Files

The most important configuration files can be found under your KonaKart installation directory at:

```
webapps\konakart\WEB-INF\classes\konakart.properties
and
webapps\konakartadmin\WEB-INF\classes\konakartadmin.properties
```

They can be edited using any text editor. Alternatively, they can be edited from the *Configuration>>Configuration Files* section of the Admin Tool. Note that most changes will not take effect until KonaKart is restarted.

Changing the Editable File List in the Admin App

In the *Configuration Files* section of the Admin Tool a list of files is shown that can be edited. By default these are the main KonaKart properties and logging files.

You can add any files you like to that list so long as they are accessible to the KonaKart server - wherever that's running in your configuration.

There's a simple XML file (called *konakart_config_files.xml*) where the files that are shown are defined. It looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<konakart-config-files>
  <file>
    <displayName>
      KonaKart Properties File
    </displayName>
    <fileName>
      ../webapps/konakart/WEB-INF/classes/konakart.properties
    </fileName>
  </file>

  <file>
    <displayName>
      KonaKart Admin Properties File
    </displayName>
    <fileName>
      ../webapps/konakartadmin/WEB-INF/classes/konakartadmin.properties
    </fileName>
  </file>

  <file>
    <displayName>
      KonaKart Logging Properties File
    </displayName>
    <fileName>
      ../webapps/konakart/WEB-INF/classes/konakart-logging.properties
    </fileName>
  </file>

  <file>
    <displayName>
      KonaKart Admin Logging Properties File
    </displayName>
    <fileName>
      ../webapps/konakartadmin/WEB-INF/classes/konakart-logging.properties
    </fileName>
  </file>
</konakart-config-files>
```

You'll find this file under *webapps/konakartadmin/WEB-INF/classes*. In the default installation on Windows that is: *C:\Program Files\KonaKart\webapps\konakartadmin\WEB-INF\classes\konakart_config_files.xml*.

You'll see that you define a path to the file and a "display name" which is the name you see in the list in the Admin Tool.

A point to note if no files are listed:

Depending on your configuration (eg. if you are not using the default tomcat container supplied in the download kit) you may find that the default definitions of the relative paths in the supplied konakart_config_files.xml will not work - you will have to amend these to suit your environment.

KonaKart Properties Files

KonaKart uses a number of properties files for configuration and by default these have the following names and locations (assuming KonaKart was installed on Linux under /home):

KonaKart engine server properties file:

/home/konakart/webapps/konakart/classes/konakart.properties

KonaKart Storefront Application client properties file:

/home/konakart/webapps/konakart/classes/konakart_app.properties

KonaKart Storefront engine AXIS (SOAP) client properties file:

/home/konakart/webapps/konakart/classes/konakart_axis_client.properties

KonaKartAdmin engine server properties file:

/home/konakart/webapps/konakartadmin/classes/konakartadmin.properties

KonaKartAdmin Admin App Client properties file:

/home/konakart/webapps/konakartadmin/classes/konakartadmin_gwt.properties

KonaKartAdmin Admin engine AXIS (SOAP) client properties file:

/home/konakart/webapps/konakartadmin/classes/konakartadmin_axis_client.properties

KonaKart engine Velocity properties file:

/home/konakart/webapps/konakart/classes/konakart_velocity.properties

KonaKart Admin engine Velocity properties file:

/home/konakart/webapps/konakartadmin/classes/konakart_velocity.properties

Using the above default properties file names is fine for most installations but if multiple versions of each are required there are ways to configure KonaKart in order to use properties files with different names and locations.

Some of the properties file names and locations can be specified as part of "EngineConfig" objects (EngineConfig for the Application Engine and AdminEngineConfig for the Admin Engine) which are specified as parameters when you create the respective engines.

If still more flexibility is required it is possible to override the properties file names and locations in a customizable java class called *PropertyFileNames.java*. *PropertyFileNames.java* is provided in every download kit and can be found under /home/konakart/custom/utils/com/konakart/util. *PropertyFileNames.java*

has a simple structure and allows you to override any properties file by modifying the `getFileName()` method. `getFileName()` takes two parameters:

`propertyFileCode` : an integer code that identifies the particular properties file (eg. `PropertyFileNames.KONAKARTADMIN_SERVER_PROPERTIES_FILE` and `PropertyFileNames.KONAKART_SERVER_PROPERTIES_FILE` are two examples)

`def`: the default value. By default this is returned for every properties file type.

Having modified the code in the `getFileName()` method it can be built using the standard custom build. The updated `PropertyFileNames.class` will be placed in the `konakart_custom_utils.jar` by the standard build as illustrated below:

```
C:\Program Files\KonaKart\custom>.\bin\kkant compile_utils, make_jar_custom_utils
Buildfile: build.xml

compile_utils:
[echo] Compile the customisable utils code
[javac] Compiling 2 source files to C:\Program Files\KonaKart\custom\utils\classes

make_jar_custom_utils:
[echo] Create the konakart_custom_utils.jar
[jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakart_custom_utils.jar

BUILD SUCCESSFUL
Total time: 1 second
```

Once built successfully the new `konakart_custom_utils.jar` should be copied to the lib directories of the konakart and konakartadmin webapps as required.

Configuration of Messages

With KonaKart you can store application messages either in the database or in files. The default is to use files (sometimes called message catalogs or message properties files).

A configuration variable (called "USE_DB_FOR_MESSAGES") defines whether the messages are stored in the database or in files. This can be set in the Admin App under Configuration >> Admin App Configuration.

Switching to Database Messages

By default, no messages will be loaded into the database. A utility called kkMessages is provided to load messages from your own message properties files. Once loaded you can switch to database messages by simply changing the "USE_DB_FOR_MESSAGES" configuration variable to "true".

Once loaded into the database the messages can be maintained on the Messages panel of the Admin App.

KKMessages Utility

A utility called "kkMessages" is provided to assist with the migration of messages from files to the database.

The kkMessages utility can be found under the utils directory wherever you installed KonaKart (from v5.2.0.0 only). Some sample scripts are provided to make the kkMessages utility easy to use. A number of functions are supported as can be seen by calling the utility with the "-?" parameter:

```
mike@luton:~/konakart/utils/kkMessages$ ./kkMessages.sh -?
=====
KonaKart Messages Utility
=====

Usage: KKMessages
  -i file           - import a file of messages
  [-en file encoding] - file encoding (default is ISO-8859-1)
  [-ce console encoding] - console encoding (default is utf-8)
  -r               - when importing first remove all existing
                      messages with the specified type and
                      locale (default is not to remove them)
  -x file           - export messages to a file
  -q file           - export messages as SQL to a file that can
                      be used to import the messages elsewhere
  -l locale         - locale (eg en_GB)
  -t (1|2|3)        - message type
                      1 = Application Message
                      2 = Admin App Message
                      3 = Admin App Help Message
  -ll              - print a list of the current languages
  -lm              - print a list of the current messages
  [-u username]     - username
  [-p password]    - password
  [-s storeId]      - storeId
  [-e (0|1|2)]      - engine mode
  [-c]              - shared customers - default is not shared
  [-ps]             - shared products - default is not shared
  [-ae adminengine classname] - default is com.konakartadmin.bl.KKAdmin
  [-?]              - shows this usage information
```

You need to load messages for each of the 3 types (type 1 (storefront application messages), 2 (Admin App messages) and 3 (Admin App Help Messages)) for every locale you wish to support. It is important that all of your language records contain a locale definition and this locale is used to define the messages for that locale. You can verify that all your languages have a locale defined by using the kkMessages utility to list your languages with the "-ll" option (see example below). You can update the locale for each language using the Languages panel of the Admin App.

A few example scripts demonstrating the usage of the kkMessages utility are provided. See kkMessages_Load_en_GB.sh, kkMessages_Load_en_GB.bat, kkMessages_Load_Defaults.sh and kkMessages_Load_Defaults.bat.

With a default installation of KonaKart where the following languages are defined:

```
mike@luton:~/konakart/utils/kkMessages$ ./kkMessages.sh -ll
=====
KonaKart Messages Utility
=====

Languages:
  ID  Name       Code   Locale
  1  English    en    en_GB
  2  Deutsch    de    de_DE
  3  Español    es    es_ES
  4  Português  pt    pt_BR
```

With a default installation you can run the kkMessages_Load_Defaults script ("sh" or "bat" versions depending on your platform) to load the 3 message types for the 4 default languages.

Logging

KonaKart uses standard log4j logging.

The KonaKart engines search for logging properties files in the following order on the classpath:

1. If not null the file defined by the System property "log4j.configuration"
2. konakart-logging.properties
3. logging.properties
4. log4j.properties

If none of the above are found on the classpath a few hard-coded properties are defined for basic logging by default.

Logging can be enabled for selected parts of the system and the level of the logging tuned as required. Use the following logging levels:

- ERROR
- WARN
- INFO
- DEBUG

"DEBUG" logging will show the most detail.

The logging properties file "konakart-logging.properties" contains a set of the most commonly-used KonaKart logging properties. Simply edit this file to modify the logging level of the subsystem you need logging for. (You will find the file under the classes directories of both the konakart and konakartadmin webapps - e.g. webapps/konakart/WEB-INF/classes)

A very common example is to set the logging to display SQL that is being executed by KonaKart. You would do this by setting the following properties:

```
# KonaKart Persistence layer - Set to INFO to see the
SQL
log4j.logger.com.konakart.db = INFO
```

Logging changes will take effect without needing to restart your application server so long as you have set the system property kk.log4j.WatchTimeSecs to a value (in seconds greater than 0). Refer to the startkonakart.sh and startkonakart.bat scripts for more details on the logging properties that control this..

Internationalization of KonaKart

KonaKart is completely multi-lingual both at the database level and at the UI level.

Translating the KonaKart Application

The data within the database such as product descriptions and category names etc. may exist in different languages. The database contains a languages table that contains information about each of the supported languages. When a new product is added to the database through the administration tool, it is possible to enter a description in multiple languages.

Languages in the database have an attribute (specifically a column called *display_only*) that indicates whether the language is only used for display purposes and not for product descriptions (and other KonaKart objects that have different variants for each data type). By default when you install KonaKart every

language that is defined in the database has this *display_only* attribute set to 0 (false) which means that all KonaCart objects that require a variant for each language must have a value defined for each language.

If you wish to introduce a language for use with the Admin App but not be required to add KonaKart object definitions for the language you can check the *Display Only* checkbox for the language (this can be found on the Languages Panel of the Admin App). You will have to provide a message catalog file for the new language and name the file in the appropriate fashion for the chosen locale.

With KonaKart you can either store the application messages in message catalog files on disk or in the database. The default is to use message catalog files but from the v5.2.0.0 release it became possible to use the database to store these messages. Some users prefer to manage the messages in the database whereas others prefer files. The choice is yours!

A configuration parameter called "USE_DB_FOR_MESSAGES" is used by the system to determine whether the messages are stored in the database or in files. You can set this value in the Admin Application under Configuration >> Admin App Configuration. The label used in English for this field is "Use D/B For Messages" and it can have a value of "true" or "false".

When the messages (the storefront Application messages, the Admin Application messages and the Admin Application Help messages) are stored in the database you should use the Messages panel of the Admin Application to maintain the messages.

KonaKart Admin Application - Messages Panel

Note that when editing the messages using the Messages Panel of the Admin Application you can click on the "Switch Editor" icon (to the right of the "Locale" field) to switch between the Rich Text Editor and the Plain Text Editor. This is sometimes useful to gain greater control over the HTML that is saved (the HTML that is created by the Rich Text Editor can be slightly different depending on which browser you use).

When the messages are stored in message catalog files, there is a message catalog for each language. These can be found in the webapps/konakart/WEB-INF/classes directory of the application server. The default message catalog is called Messages.properties and is used if the language specific catalog isn't found.

The naming convention of the message catalogs is as follows:

```
// contains all the storefront strings  
Messages_[language-code].properties
```

For [language_code] you should use the 2-character language code that you have set up in the languages section of the admin application. For example, you might have "fr" for French, "zh" for Chinese, "de" for German, "ja" for Japanese etc. You can also use the full locale if you prefer (such as "en_UK", "es_ES", "pt_BR" etc).

In order to change the language of the storefront application the current customer clicks a link which runs the code in *SetLocaleAction.java*

Translating the country names

The KonaKart database allows for only one definition of a country name. In some cases (where only one language is used for the store) this is sufficient and no translation of these names is required. In cases where it's required to show the country names in the language of the supported locales you need to follow the following instructions.

You can display a list of countries (for example for display on the customer registration form) in the language of the currently-selected locale by taking the following steps:

- Set the translated country names in the message catalog for each supported locale (or set these values in the database if you use the database for your messages). The key to use is defined for each country in the countries table of the database (as an example, the default key for the "United States" is "CTRY.USA").
- Enable the lookup of the translated messages by setting the *Use Country Names in Msg Cat* configuration variable to true. (The configuration variable key is *USE_MSG_CAT_FOR_COUNTRY_NAMES*). You can find this configuration setting on the *Configuration >> Store Configuration* tab of the Admin Application.

Once set up you can use a call on the KonaKart Application Client Engine (KKAppEng) called getAllCountries() to get a list of countries with the translated names. The list of country names returned is ordered according to the collation rules of the selected locale. You can see examples of the use of this call in the RegisterCustomerBody.jsp source file provided in the download kits.

Translating the KonaKart Admin Application

The KonaKart Administration Application can be completely translated by either editing the messages in two message catalog files (for file-based messages) or by editing the messages stored in the database using the Admin Application itself.

To make the Admin Application available in a new language you need to supply two new message catalog files (for file-based messages) or simply add a complete set of database records for the new language (when KonaKart is configured for database-based messages).

Whichever language you use this is also a handy way to re-label the "custom" fields that appear on many of the important objects within KonaKart to reflect the meaning in your particular system.

In the case of file-based messages, your new message catalogs must be called:

```
// contains all the strings except the help page text  
AdminMessages_[language-code].properties  
  
// contains just the text on the help pages  
AdminHelpMessages_[language-code].properties
```

For [language_code] you should use the 2-character language code that you have set up in the languages section of the admin application. For example, you might have "fr" for French, "zh" for Chinese, "de" for German, "ja" for Japanese etc. You can also use the full locale if you prefer (such as "en_UK", "es_ES", "pt_BR" etc).

For file-based messages you should place your completed message catalogs in this directory:

webapps/konakartadmin/WEB-INF/classes

Since there are quite a large number of messages to translate, you might choose to do this over a period of time. A recommended approach is to start your new message catalogs with copies of the default (English) catalogs (called AdminMessages.properties and AdminHelpMessages.properties) then translate the messages that are most important to you first and complete the rest as time permits.

You can use a "substitution" syntax in AdminHelpMessages.properties if you wish to create messages that are too large for the database to store (4000 bytes in Oracle and DB2). The substitution syntax is like the UNIX shell variable format: \${variable}. An example of the technique is provided for the help.editProductDetails property in AdminHelpMessages.properties where it is defined as the concatenation of two other properties (help.editProductDetails_1 and help.editProductDetails_2) that are also declared in AdminHelpMessages.properties.

There are also a number of strings in the database that need to be translated. These are used by the Admin App to label some of the configuration parameters and provide helpful comments. If you issue a: "SELECT configuration_title, configuration_description FROM configuration;" SQL query, you will see the values that you can update.

Also, you may wish to include translations for the velocity templates (eg. EmailNewPassword, OrderDetails, OrderInvoice, OrderPackingList and OrderStatusChange).

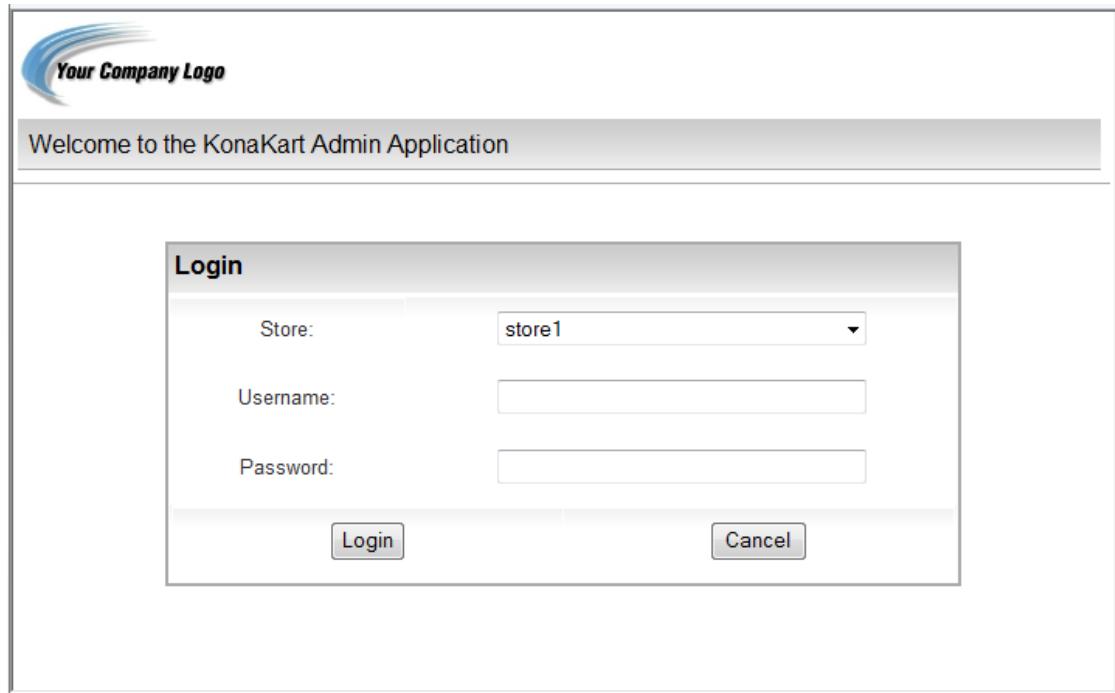
Once you have completed the message catalogs, velocity templates and a sql update script for your language, please contribute these to the KonaKart Community by posting them to the contributions section of our forum for the benefit of other users.

Changing the Logo on the KonaKart Admin Application

This has been made a lot easier from version 5.4.0.0 where you can use your own logo instead of the default KonaKart logo by simply replacing the file that contains the KonaKart logo directly under the

konakartadmin webapp (the file is approximately 6KB in size). You must use a jpg file and the dimensions should ideally be the same as the KonaKart Logo which are: 240x60 pixels.

For example you can replace the KonaKart logo with your own logo as illustrated below:



KonaKart Admin Application - Custom Logo

Changing the Date Format in the KonaKart Application

The templates are stored in the message catalog:

```
# -- Date formats --
date.format=dd/MM/yyyy
date.time.format=dd/MM/yyyy HH:mm
# jquery datepicker has a different formatter to standard java
datepicker.date.format=dd/mm/yy
```

date.format and *date.time.format* are used for formatting dates within the storefront application. *datepicker.date.format* is used to format the date received from the jQuery date picker widget.

Formatting of Addresses

In the admin app under Localizations >> Address Formats, there are a number of templates which can be associated to countries, in order to format addresses correctly for the country. Valid template key words are:

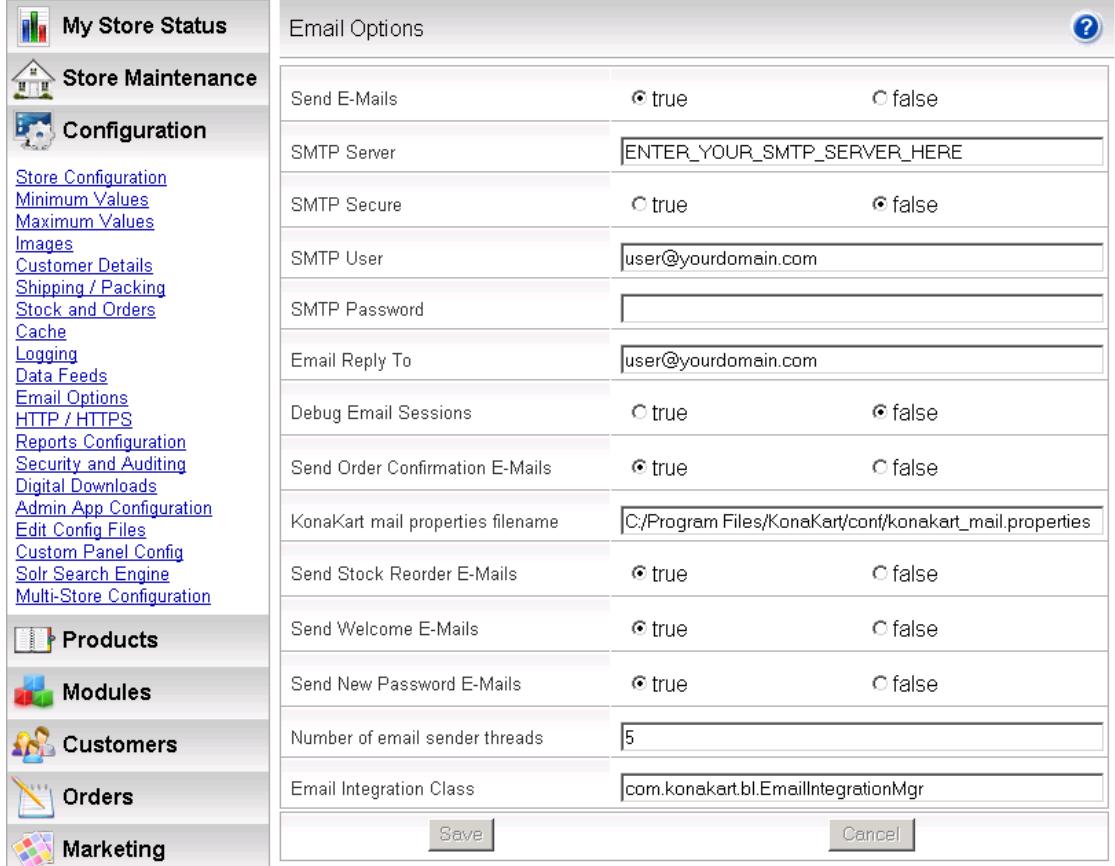
- \$cr - Carriage return (new line)
- \$firstname

- \$lastname
- \$company
- \$streets = \$street + line break + \$suburb
- \$street
- \$street1
- \$suburb
- \$city
- \$postcode
- \$state
- \$telephone
- \$telephone1
- \$email
- \$country

The formatting is performed by substituting the above keywords with the actual values within the address object. A typical template could be : \$company\$cr\$firstname \$lastname\$cr\$streets\$cr\$city, \$postcode\$cr \$state, \$country\$crtel: \$telephone\$crtel1: \$telephone1\$creMail: \$email .

Email Configuration

In most cases email can be configured completely within the KonaKart Admin App in the *Configuration>>Email Options* section (see image below). Here you can set up your SMTP server host, your SMTP username, SMTP password and whether or not the SMTP server requires authentication:



The screenshot shows the KonaKart Admin Application interface. On the left is a sidebar with various menu items: My Store Status, Store Maintenance, Configuration, Products, Modules, Customers, Orders, and Marketing. The Configuration item is currently selected. The main panel is titled "Email Options". It contains several configuration fields with radio button options (true/false) and input fields:

Send E-Mails	<input checked="" type="radio"/> true	<input type="radio"/> false
SMTP Server	ENTER_YOUR_SMTP_SERVER_HERE	
SMTP Secure	<input checked="" type="radio"/> true	<input type="radio"/> false
SMTP User	user@yourdomain.com	
SMTP Password		
Email Reply To	user@yourdomain.com	
Debug Email Sessions	<input checked="" type="radio"/> true	<input type="radio"/> false
Send Order Confirmation E-Mails	<input checked="" type="radio"/> true	<input type="radio"/> false
KonaKart mail properties filename	C:/Program Files/KonaKart/conf/konakart_mail.properties	
Send Stock Reorder E-Mails	<input checked="" type="radio"/> true	<input type="radio"/> false
Send Welcome E-Mails	<input checked="" type="radio"/> true	<input type="radio"/> false
Send New Password E-Mails	<input checked="" type="radio"/> true	<input type="radio"/> false
Number of email sender threads	5	
Email Integration Class	com.konakart.bl.EmailIntegrationMgr	

At the bottom are "Save" and "Cancel" buttons.

KonaKart Admin Application - Email Configuration

As with all fields in the Admin App, use the floatover text on each label to find out more about each configuration option.

For more advanced users there is a *konakart_mail.properties* file in which you can define additional mail properties. (In fact, you can rename this file if you wish and change its location - so long as you define the filename holding these values in the Admin App - see "KonaKart mail properties filename in the image above)). A common example of the use of this *konakart_mail.properties* file is for setting up access to the Google Mail SMTP server (this uses a non-standard port (465), requires authentication and uses encryption. An example of the parameters to set to use the Google SMTP mail gateway is provided in the default *konakart_mail.properties* file.

Modifying the Email Templates

If configured to do so, KonaKart will send out emails after registration, to confirm orders and to distribute new passwords. The emails are generated using Velocity [<http://velocity.apache.org>] templates which can be found under the templates directory under the installation home. The current templates are all files following the pattern *.vm. Note that the file name must end with an underscore followed by a two letter country code (i.e. OrderConfirmation_en.vm).

When the state of an order is changed through the Admin App, an eMail may be sent to the customer based on the template called *OrderStatusChange_xx.vm* where xx is the two letter country code. A different template may be defined for each order state where the naming convention is *OrderStatusChange_stateId_xx.vm*. For example if the order changes state from state 1 to state 2, it will use the template called *OrderStatusChange_2_xx.vm* if it exists. If it doesn't exist, KonaKart will use the default *OrderStatusChange_xx.vm* template.

Adding Custom Business Objects for use in Velocity Templates

In the Enterprise version it is possible to use your own business objects in your Velocity templates. To add your own business objects to the Velocity context you must create a class that implements *com.konakart.blif.VelocityContextMgrIf* (similarly for the Admin version at *com.konakartadmin.blif.AdminVelocityContextMgrIf*).

An example of adding an object to the Velocity context is provided in the default implementation which can be found at "*custom\appnEE\src\com\konakart\bl\VelocityContextMgr.java*" (under the installation home directory).

You can modify the above source file (and the equivalent one for the admin engine at "*custom\adminappnEE\src\com\konakartadmin\bl\AdminVelocityContextMgr.java*" as required), then execute the ANT build file under the custom directory to create a new *konakart_customEE.jar* (or *konakartadmin_customEE.jar* as applicable) containing your new implementation.

Once your modifications to the VelocityContextMgrs are built you can then reference your own business objects from your Velocity templates.

Search Engine Optimization (SEO) Features

The SEO features can be configured through the message catalogs with the attributes beginning with "seo." (e.g seo.default.meta.description or seo.meta.keywords.template). The catalogs contain multi-lingual templates in order to write product information (name, model, manufacturer, category) into:

- The URL
- The window title
- The meta description
- The meta keywords

The templates may contain the following placeholders : \$category, \$manufacturer, \$name, \$model which are substituted by real data depending on what is being viewed at the time. Each attribute within the message catalog has a description which explains how it is used.

From version 5.7.5.0 the URL formatting is configurable. The default formatting creates a URL with a tree like structure:

/DVD-Movies/Action/Warner/Fire-Down-Below/DVD-FDBL/2_11.action

The other option, selectable from the Admin App (Configuration >> Store Configuration) adds parameters to the URL:

SelectProd.action?prodId=11&manufacturer=Warner&category=Action&name=Fire+Down+Below&model=DVD-FDBL

The source of the code that reformats the URLs is available in the Struts actions of the storefront application.

Sitemap Generation

The Enterprise version of KonaKart contains a batch program to generate an XML sitemap. Sitemaps are an easy way for to inform search engines about pages that are available for crawling.

The program class is called `createSitemapBatch` and can be found in `KonaKart\custom\batch\src\com\konakartadmin\bl\AdminProductBatchMgr.java`. It may be scheduled to run at regular intervals using Quartz or may be run interactively from the Admin App by clicking on *Scheduler* on the menu bar.

A number of files are produced by the batch program:

- `sitemap.xml` - the sitemap index file that includes the other sitemap files
- `sitemap-products_n.xml` - includes links to all of the product detail pages. Depending on the number of products in the database, multiple files may be produced named `sitemap-products_1.xml`, `sitemap-products_2.xml` etc.
- `sitemap-categories.xml` - includes links to all of the top level category landing pages.
- `sitemap-pages.xml` - includes the home page and various static pages

You may need to edit the program in order to include more or remove some of the static pages depending on the content of your eCommerce storefront application.

The files are created in a directory defined by a configuration variable which can be set through the *Configuration >> Sitemap Configuration* panel of the Admin App. If you need to modify the batch program and to add more configurable parameters, they may be added to this panel. In order to be picked up by a search engine the files must be placed under the root directory of the storefront; e.g. <http://www.kkstore.com/sitemap.xml>. If the batch file cannot reach this location, you may still automate the process by setting up a cron job or using some other utility to move the generated files. More information on sitemaps may be found at <http://www.sitemaps.org>.

If you need to make modifications to the sitemap generation program such as adding or removing static pages, modifying priority of entries or the URL change frequency, you may modify the source code and recompile.

Caching

The Enterprise version of KonaKart provides functionality to configure caches for a growing number of KonaKart objects.

By default the Open Source Ehcache cache is used which can be configured in the `konakart_ehcache.xml` configuration file. This file is located in the `webapps/konakart/WEB-INF/classes` directory of a standard installation.

Three cache types are currently available with different characteristics:

- Ehcache - the default cache. This allows very flexible configuration in `konakart_ehcache.xml` and stores serialized objects on the JVM heap.
- Big Memory Go - the Terracotta cache. This allows very flexible configuration in `konakart_ehcache.xml`, stores serialized objects and can store the cache off the JVM heap thereby utilising up to 32Gb of memory space. This gives you an opportunity to store very large caches in memory

without being restricted by the standard JVM heap limit on your platform. To use the free version of the Big Memory Go cache you need to obtain a license key from Terracotta (see below for details).

- KonaKart cache - this is the highest performance cache since it stores java objects so no serialization/de-serialization is required. The disadvantages are that there are no configuration or monitoring options and the data must be held on the JVM heap (so there are restrictions in how much data can be stored).

Currently there are two caches supported but the number is likely to grow in the future as more opportunities for improving performance are implemented:

- Product Cache - a cache of complete products
- Product Image Names Cache - a cache for storing the set of image names that have been defined for a product.

In the *Configuration >> Cache* panel in the Administration Application you can choose to enable or disable each of the above two caches.

To use the KonaKart cache simply remove the konakart_ehcache.xml file (if that file is found on the classpath an Ehcache cache will be used).

To use a Big Memory Go cache you need to take the following steps:

- Follow the instructions here: <http://terracotta.org/documentation/4.0/bigmemorygo/get-started>
- Replace the ehcache jar used in KonaKart with the ehcache-ee-* jar in BigMemoryGo.
- Add the bigmemory-* jar to the konakart webapp lib directory
- Obtain a license key file for using BigMemory Go and add it to the classpath (eg. add it to webapps/konakart/WEB-INF/classes)
- Configure the konakart_ehcache.xml file according to your requirements (eg add Off Heap caching) EG:

```
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="ehcache.xsd"
  name="KonaKart-Ehcache"
  updateCheck="true"
  monitoring="off"
  dynamicConfig="false"
  maxBytesLocalHeap="1M"
  maxBytesLocalOffHeap="2G">
```

- Add -XX:MaxDirectMemorySize=4G to your JVM initialisation parameters (or equivalent setting for your platform).
- Use the BigMemory Go Management Console to manage the Cache if required.

If you need more control over which products are cached and which are not you can create your own version of the ProductMgr and override the cacheThisProduct product. This method takes a product as an argument and returns true if the product should be cached and false if not. An example of such a ProductMgr class is provided at "java_api_examples\src\com\konakart\apiexamples\MyProductMgrEE.java".

Adding Custom Functionality to the Admin App

From version 2.2.6.0 the Admin App allows you to define custom panels and custom buttons in defined areas, to allow you to extend the Admin App by adding your own administration functionality outside the standard Admin App.

Adding Panels

There are two types of custom panels that can be added. One type is set in the Custom Section of the Administration Application, the other type allows you to set them in the other sections (e.g. "products", "orders" etc).

In total there are 20 custom panels that display a predefined URL in a frame. The 10 URLs that are used for the Custom Panels in the Custom Section of the Administration Application are defined in the *Configuration>>Custom Panel Config* section of the Admin App. The 10 URLs for the "floating" Custom Panels are defined in the *konakartadmin_gwt.properties* file in "File-Based Configuration".

More details on how to configure the "floating" Custom Panels can be found in the *konakartadmin_gwt.properties* file.

In all cases the labels of the panels can be changed by editing the message catalog. The panel links can be rendered visible / invisible by configuring the role based security. The session id of the logged in user is appended to the URL so that the custom application can control security using the Admin Engine API. The string appended to each url is similar to the following, except that the session id will be random:

sess=6596bd465a824bb9cd9a6080af07e02f

With the Enterprise version a few extra parameters are added:

- langId - Currently selected display language Id
- panelName - Name of the panel
- userId - Id of the logged in Admin User
- storeId - Id of the current store
- mode - Engine mode
- cush - Customers Shared
- prsh - Products Shared
- cash - Categories Shared

An example for the Enterprise version is:

- sess=6596bd465a824bb9cd9a6080af07e02f
- &langId=1
- &panelName=kk_panel_customE
- &userId=2
- &storeId=storeX
- &mode=2
- &cush=true
- &prsh=true
- &cash=false

You can also add your own parameters to the URLs that you define. For example:

<http://www.mycustom.com/custom1.action?parm1=abc>

Each custom panel may have its own online help. The text for the online help is defined in the file *AdminHelpMessages.properties* under the keys, help.customPanel1 to help.customPanel10.

You can set the height of the custom panel frame to suit your needs by specifying a number of pixels (e.g. 700px).

Adding Custom Configuration Panel

A "User Defined" panel is available to use for maintaining custom or user defined configuration parameters that are specific to your own store system.

All you need to do is to insert configuration parameters into the configurations table in your KonaKart database in configuration group 31 and they will appear automatically on the User Defined Configurations panel of the Administration Application.

As a guide to inserting configuration parameters refer to the values already in the configurations table and or take a look at the *konakart_demo.sql* file (under the database directory in your KonaKart installation) for many example SQL statements for creating configuration parameters.

Adding Buttons

The buttons will call pre-defined urls and will normally pass some parameters to allow you to check security and/or operate on selected objects. The button labels are defined in the Configuration>>Admin App Configuration section of the Admin App and the buttons remain invisible until the button labels are set with some text. The buttons and parameters available are in the following panels:

- Customer Panel
 - *id* - the customer id
 - *sess* - the session id of the admin user
- Returned Products Panel
 - *order_return_id* - id of the OrderReturn object
 - *order_id* - id of the Order object
 - *total_inc_tax* - total amount of the Order
 - *sess* - session id
 - *tx_id* - gateway transaction

Adding A Custom Application - Insert Product Wizard

The Enterprise version of KK contains an Insert Product Wizard to illustrate an example of how a custom panel may be used to provide functionality directly tailored for your unique business requirements. The wizard consists of a separate application which can be found in the *konakartadmin_app* directory under *webapps*.

It's a simple application that allows you to insert a product in four easy wizard driven steps. This example was chosen because the Insert / Edit Product panel of the KonaKart Admin App can be rather daunting for non-technical users since it allows you to configure all aspects of a product. Most customers don't use

all of the product features and many desire a very simple insert product panel that can be used by non-technical staff and that displays only the required attributes.

The application uses the same Struts2 / JSP technology used by the storefront application. In reality all of the functionality is displayed by a single JSP called InsertProduct.jsp which displays the wizard. There are no page refreshes. The wizard tabs are populated using AJAX calls and the form validation uses jquery validate. The wizard steps are:

- Insert product details
- Select a manufacturer
- Select one or more categories
- Select the product images

1. Product Details 2. Manufacturer 3. Categories 4. Images

Enter the product details and then click the "next" button.

Product Name	Product Price
<input type="text" value="My Test Product"/>	<input style="background-color: #ffcccc; border: 1px solid red;" type="text"/>
This field is required.	
Product Description	
This field is required.	

[Previous](#) [Next](#)

The controlling JavaScript is in a file called kk.admin.js in the script directory.

The example should provide a good starting point for you to build your own custom panel implementing whatever functionality your business requires.

Searching with wildcards

In version 5.0.0.0 of KonaCart, the search constraints for most objects are paired with a search rule that can be configured in order to determine whether a wildcard is added before, after or before and after

the search string. These "rule" attributes can be set when making the API call that searches for products, orders, customers etc.

The admin application has a properties file called *konakartadmin_gwt.properties* which is used to configure the search rules for the various panels of the Admin App.

```
# -----
# Default wild card settings for the Admin GWT Application
# -----
#
# Allows you to set wild card settings when searching
# for data using the Admin App. The accepted values are:
#
#     SEARCH_EXACT = 0;
#     SEARCH_ADD_WILDCARD_BEFORE = 1;
#     SEARCH_ADD_WILDCARD_AFTER = 2;
#     SEARCH_ADD_WILDCARD_BEFORE_AND_AFTER = 3;
#     SEARCH_USE_GLOBAL_CONFIG = 4;
#
# When set to 4, the search parameters will use the global settings
# defined by the configuration variables: ADMIN_APP_ADD_WILDCARD_BEFORE
# and ADMIN_APP_ADD_WILDCARD_AFTER. The global settings will also be used if
# this file is not found.
# If not set the default value is 4 (SEARCH_USE_GLOBAL_CONFIG).
#-----

#sr.address.company.name      = 4
#sr.address.format           = 4
#sr.address.summary.format   = 4
#sr.catalog.name              = 4
#sr.country.iso2              = 4
#sr.country.iso3              = 4
#sr.country.name              = 4
#sr.coupon.code              = 4
#sr.coupon.name              = 4
#sr.currency.name             = 4
#sr.customer.email             = 4
#sr.customer.city              = 4
#sr.customer.first.name       = 4
#sr.customer.last.name        = 4
#sr.customer.postcode         = 4
#sr.customer.street           = 4
etc. etc.
```

Note: Prior to v6.6.0.0 these parameters were defined in a file called *AdminSearchRules.properties* which is no longer used.

As can be seen in the example above, there are various rules such as SEARCH_EXACT and SEARCH_ADD_WILDCARD_BEFORE etc. which map to integer values. If for example I want to be able to search for customers using the customer last name with a trailing wildcard, I must set:

sr.customer.last.name = 2

Before version 5.0.0.0, there were global configuration variables for the admin app called:

- ADMIN_APP_ADD_WILDCARD_BEFORE
- ADMIN_APP_ADD_WILDCARD_AFTER

These variables still exist and will be used when the rules are set to the value SEARCH_USE_GLOBAL_CONFIG, which is the default case for backwards compatibility.

Case In-Sensitive Searching

From version 6.6.0.0 of KonaKart it is possible to configure case-insensitive searching, for selected attributes, for databases that don't support case insensitive searching by default (such as PostgreSQL, Oracle and DB2). For the Admin App, the case-sensitivity configuration is defined using the File-Based Configuration mechanism provided in the Enterprise version of KonaKart.

These particular settings make no difference for databases that use case insensitive searching by default (such as MySQL).

The File-Based Configuration is set in the properties file called *konakartadmin_gwt.properties* as follows:

```
#-----
# Use this to define which search fields case should be ignored
# (default is false = case is not ignored)

fbc.kk_panel_customers.first_name_ignore_case          = true
fbc.kk_panel_customers.last_name_ignore_case          = true
fbc.kk_panel_customers.email_address_ignore_case      = true
fbc.kk_panel_customers.street_address_ignore_case     = true
fbc.kk_panel_customers.street_address1_ignore_case    = true
fbc.kk_panel_customers.postcode_ignore_case           = true
fbc.kk_panel_customers.city_ignore_case                = true

fbc.kk_panel_products.search_ignore_case              = true
fbc.kk_panel_products.sku_ignore_case                 = true
```

You can choose to set "ignore case" on all of the attributes supported or just a subset of them.

Both Community Edition and Enterprise Editions support this case insensitive search possibility through the APIs because the AdminProductSearch and AdminCustomerSearch have attributes that define the case-insensitivity for the respective attributes.

Before setting these particular "ignore case" settings please pay attention to the impact on performance that might occur. You are most likely to get better performance if you do not set "ignore Case" to "true".

There are other techniques to achieve case-insensitive searching in those databases that don't support this by default. Refer to database-specific instructions if you choose to take this alternative option.

For example, with DB2 you can define the case sensitivity behaviour when ordering results by setting the database collating sequence. In Oracle you can set case-sensitivity using the NLS_COMP and NLS_SORT configuration variables.

Making something happen when a product needs to be reordered

When the number of products in stock hits the reorder level defined by the configuration property STOCK_REORDER_LEVEL, KonaKart instantiates a class defined by the property STOCK_REORDER_CLASS. If this property isn't set, the class that is instantiated is com.konakart.bl.ReorderMgr. If you write a custom class it must implement the interface com.konakart.bl.ReorderMgrInterface which contains only one method:

```
public void reorder(int productId, String sku, int currentQuantity) throws Exception;
```

After the class is instantiated, the reorder method is called and information regarding the productId, the SKU and current product quantity is passed to the method so that it can use this information to trigger an event. This mechanism is a useful generic way to interface KonaKart to external systems since the custom class could write data to a database, call a web service or write data to a file etc.

If the configuration variable STOCK_REORDER_EMAIL is set, then KonaKart will also send an eMail alert using the *LowStockAlert* template.

All of the configuration variables can be edited in the *Configuration>>Stock and Orders* section of the Admin App.

Making something happen when the state of an order changes

When an order is saved in the database or the status of an order is changed (i.e. through the callback of a payment gateway), KonaKart instantiates a class defined by the property ORDER_INTEGRATION_CLASS. If this property isn't set, the class that is instantiated is com.konakart.bl.OrderIntegrationMgr. If you write a custom class it must implement the interface com.konakart.bl.OrderIntegrationMgrInterface which contains some methods:

```

/**
 * Called just after an order has been saved. The order details are passed to the method so that
 * all the information should be available in order to integrate with external systems.
 *
 * @param order
 */
public void saveOrder(OrderIf order);

/**
 * Called just before an order has been saved. This method gives the opportunity to modify any
 * detail of the order before it is saved. If null is returned, then no action is taken. If a
 * non null Order is returned, then this is the order that will be saved.
 *
 * @param order
 * @return Returns an order object which will be saved
 */
public OrderIf beforeSaveOrder(OrderIf order);

/**
 * Called just after an order status change
 *
 * @param orderId
 * @param currentStatus
 * @param newStatus
 */
public void changeOrderStatus(int orderId, int currentStatus, int newStatus);

/**
 * This method allows you to introduce a proprietary algorithm for creating the order number for
 * an order just before the order is saved. It is called by the saveOrder() method.
 * The value returned by this method populates the orderNumber attribute of the
 * order when it is saved.
 * If a null value is returned, then the order number of the order is left unchanged.
 * If an exception is thrown, the exception will be also thrown by the saveOrder() method and
 * the order will not be saved.
 *
 * @param order
 * @return Return the order number for the new order
 * @throws Exception
 */
public String createOrderNumber(OrderIf order) throws Exception;

/**
 * This method allows you to generate a tracking number for an order just before the order is
 * saved. It is called by the saveOrder() method. The value returned by this method
 * populates the trackingNumber attribute of the order when it is saved.
 * If a null value is returned, then the tracking number of the order is left unchanged.
 * If an exception is thrown, the exception will be also thrown by the saveOrder() method and
 * the order will not be saved.
 *
 * @param order
 * @return Return the tracking number for the new order
 * @throws Exception
 */
public String createTrackingNumber(OrderIf order) throws Exception;

/**
 * Called just before a subscription is inserted. This method gives the opportunity to modify
 * any detail of the subscription before it is inserted. If null is returned, then no action is
 * taken. If a non null subscription is returned, then this is the subscription that will be
 * saved.
 *
 * @param subscription
 * @return Returns a Subscription
 * @throws Exception
 */
public SubscriptionIf beforeInsertSubscription(SubscriptionIf subscription) throws Exception;

and more.....

```

The state of an order may be changed manually through the Admin App. In this case KonaKart instantiates a class defined by the property ADMIN_ORDER_INTEGRATION_CLASS. If this property isn't set,

the class that is instantiated is com.konakartadmin.bl.AdminOrderIntegrationMgr. If you write a custom class it must implement the interface com.konakartadmin.bl.AdminOrderIntegrationMgrInterface which contains the method:

```

/**
 * Called just after an order status change
 *
 * @param orderId
 *         Id of the order
 * @param currentStatus
 *         Current state of the order
 * @param newStatus
 *         New state of the order
 */
public void changeOrderStatus(int orderId, int currentStatus, int newStatus);

/**
 * This method may be customized in order to implement an algorithm that creates an RMA code for
 * the order. The Administration Application will use the returned value (if not null) to
 * automatically populate the RMA code entry field.
 *
 * @param orderId
 *         Id of the order
 * @return Returns an RMA Code
 */
public String getRMACode(int orderId);

```

This mechanism is a useful generic way to interface KonaKart to external systems since the custom classes could write data to a database, call a web service or write data to a file etc.

The ORDER_INTEGRATION_CLASS and ADMIN_ORDER_INTEGRATION_CLASS properties can be edited in the Configuration>>Stock and Orders section of the Admin App.

PDF Invoices

PDF invoices can be created by KonaKart for a variety of different purposes including:

- To send to customers as email attachments in order confirmation emails.
- For internal record-keeping - some store owners may wish to save all their invoices in PDF format.
- For display and download from both the store front (under the customer's "My Account" page) and the Admin Application. In order to activate the download links, you must set the "Enable" configuration variable in the *Configuration >> PDF Configuration* section of the Admin App.

By default, no PDF invoices are created by KonaKart. They can be created in a number of different ways:

- The application code can be modified to create the PDF invoices typically at the point when an order is confirmed or when payment is received. The methods that need to be modified are getEmailOptions() in CheckoutConfirmationSubmitAction.java (for when an order is confirmed) and sendOrderConfirmationMail() in BaseGatewayAction.java (for when payment is received).
- A batch job is provided that can be executed from Quartz (using standard KonaKart job scheduling) that will create invoices for orders that haven't yet had invoices created. This batch job can be modified to suit your particular requirements - for example you may only want invoices to be created when an order reaches a certain order status. Full source code is provided for the batch jobs to enable you to make any modifications required. By default the batch job is not scheduled to execute. To schedule it to run you need to uncomment the cron-expression tag for the CreateInvoicesTrigger in your konakart_jobs.xml Quartz job definition file.

- Invoices can be created by calling a "createInvoice()" method from the OrderIntegrationMgr and/or the AdminOrderIntegrationMgr. This call is commented out in the supplied versions of these two order integration managers making it easy to modify to create the invoices at this stage. As in the batch job, it would also be easy to code logic here that only created invoices when the order reached a certain specified order status if that was required.
- Invoices can be created "on-the-fly" by calling the GetPDF servlet. This is useful when you don't ever want to create a permanent record of the PDF file on the file system (possibly for Data Protection restrictions). The GetPDF servlet will return the PDF invoice if it already exists or, if it doesn't exist, create the PDF invoice dynamically and then return the contents of this.

By default, KonaKart stores the created invoices in a hierarchical structure on the file system. The directory structure is deep to distribute the files evenly across multiple directories to avoid the possibility of creating too many files in a single directory. The root location is defined in the "PDF_BASE_DIRECTORY" configuration variable. This can be set to any location accessible to the KonaKart code that creates the invoices.

An example PDF invoice filename, on Linux, is: "/home/greg/konakart/pdf/store1/2/2010/3/199-547b759d-735c-48bb-a23d-4cf526be9c3a.pdf"

In turn, the directories used are:

- "store1" - the storeId
- "2" - 2 is the code for an Invoice (other document types are packing slips (3) and order details (1))
- "2010" - the year at the time the invoice was created (YYYY)
- "3" - the month at the time the invoice was created (M)
- "199-547b759d-735c-48bb-a23d-4cf526be9c3a.pdf" - the filename has the format: orderId + "-" + UUID + ".pdf"

Activating a Promotion

You must follow these steps :

- Ensure that the promotion module that you want to use, is installed. All promotion modules can be found in the *Modules>>Order Total* section of the Admin App.
- Create a new promotion in the *Products>>Promotions* section of the Admin App. The online help provides extra information and most attributes have floatover text with a more detailed description. When creating the promotion, you must select which promotion module to use from the *Promotion Type* drop list. The configuration data required, depends on the promotion type. Once the promotion has been saved, it is immediately active as long as the *Active* check box was ticked and the current date lies between the promotion start and end dates.
- Rules may be added to the promotion by clicking on the *Rules* button. When a rule is selected, a pop-up window opens where the rule can be configured (e.g. Activate promotion for products belonging to a certain category). When the pop-up window is closed, the rule is saved and the *Promotion Rules* window shows a summary of the rules that have been set.
- If the promotion needs to be activated through a coupon, you can create a coupon by clicking the *Coupons* button. The coupon must be given a name and a code (the text entered by the customer to activate the coupon). The coupon may also be programmed so that it can only be used a defined number

of times. The final step is to enable the coupon entry field in the UI during checkout. To do this you must set the "Display Coupon Entry Field" to true in the *Configuration>>My Store* section of the Admin App.

Applying Promotions to Products

In KonaKart a promotion is calculated by an OrderTotal module. When an order is created, the promotion module calculates the value of the promotion discount considering all of the data contained in the order and generates a promotion line item (OrderTotal) for the order containing the discount.

Sometimes it is convenient to calculate and display the promotion discount for an array of products without requiring that the customer has added the products to the cart. In this way the customer may see the discount and be tempted to add the products to the cart.

A KonaKart OrderTotal module has a method that may be implemented to calculate a discount for a single product. The method is called:

PromotionResult getPromotionResult(Product product, Promotion promotion)

An example of its implementation may be seen in the ProductDiscount promotion module which can be found under the `custom\modules\src\com\konakart\bl\modules\ordertotal\productdiscount` directory. The method creates and returns a `PromotionResult` object which contains information to identify the promotion, the value of the promotion, whether it is a percentage or value discount and other information. The `PromotionResult` is attached to the product by the KonaKart engine. If the product is configurable (i.e. size, color etc.) each configuration may have its own array of `PromotionResult` objects enabling you to discount for example a blue shirt but not a red one.

The API call that evaluates the promotions for an array of products (Enterprise Only) is:

ProductIf[] getPromotionsPerProducts(String sessionId, int customerId, ProductIf[] products, PromotionIf[] promotions, String[] couponCodes, PromotionOptionsIf options)

A full description can be found in the Javadoc. The method receives an array of products and returns an array of products with attached `PromotionResult` objects that contain the results of one or more of the promotions passed in as a parameter. A list of active promotions can be fetched using the following API call:

PromotionIf[] getAllPromotions()

In the KonaKart demo storefront application, the promotions are cached in order not to retrieve them every time since this would degrade performance. The promotions may also be filtered and only a subset sent to the API call to calculate the discount.

An example of this promotion functionality may be found in `CatalogMainPageAction.java`. The code (which is commented out) passes the array of new products to the API call in order to determine whether any promotions apply to the products. The JSP called `NewProductsWithDetailBody.jsp` displays the promotion name next to the product image if the product has a promotion.

So to summarize, to activate the promotion for products functionality in our demo storefront application for the display of the "Latest Products", you need to take the following steps:

- Using the Admin App, create a `ProductDiscount` promotion and activate the `ProductDiscount` Order Total Module.
- In `CatalogMainPageAction.java`, uncomment the code that makes the `getPromotionsPerProducts()` API call and compile the action class.

- At this point, the Latest Products tile on the home page of the storefront should display the promotion name next to each product if the promotion is active. You may want to modify this to display an image showing the promotion details.
- In order to activate this functionality in other parts of the storefront app, you will need to add code in other struts action classes similar to the code in CatalogMainPageAction.java.

Displaying Coupon Entry Fields in your Store

You must set the "Display Coupon Entry Field" to *true* in the *Configuration>>My Store* section of the Admin App.

Configuring Digital Downloads

KonaCart supports digital downloads which can be downloaded from the KonaCart online store, as soon as payment is received for the products. A new section appears on the customer's Account page, with a link for each downloadable product in the order.

To configure KonaCart for selling digital downloads, the following steps must be taken :

- Using the Admin App, ensure that the "Product Type" attribute of all downloadable products is set to "Digital Download". You must also enter the "File Path" and the "Content Type"
 - File Path : The File Path will be concatenated to the "Base Path" defined by a configuration variable. For example, if the Base Path is set to "C:/images/" and the File Path is set to "cities/london.jpg", then the downloaded file will be C:/images/cities/london.jpg . If the Base Path is left empty (default value) then the File Path must be set to the full name, i.e. C:/images/cities/london.jpg . Note that if the Base Path ends with a "/" then the file path should not start with a "/" because the two strings are concatenated to create the full path for the digital product.
 - Content Type : The Content Type describes the content of the file. Examples are "image/jpeg" or "application/pdf". A list of Content Types may be found [here](#).
- Using the Admin App in the section Configuration>>Configure Digital Downloads you may configure some parameters regarding the digital download functionality. These are:
 - Base Path : The base path of the downloadable files as explained above.
 - Max Number of Downloads : The maximum number of times that the file can be downloaded. -1 for an unlimited number of downloads.
 - Number of days link is valid : The number of days that the download link is valid for. -1 for an unlimited number of days.
 - Delete record on expiration : When the download link expires, the database table record defining the link will be automatically deleted. Setting this to true avoids having to do manual cleanup of the database.
 - Download as an attachment : When set to true, the digital download is downloaded as an attached file that can be saved on disk. When set to false, the browser attempts to display or run the file.
- The digital downloads shipping module must be installed in the Modules>>Shipping section of the Admin App. This shipping module activates automatically when the whole of the order consists of digital downloads and displays a \$0 shipping cost. Any other shipping module should not return a shipping quote in the case of a digital download order.

Configuring Bookable Products

The Enterprise version of KonaKart supports bookable products. A bookable product is a product that has a start and end time and may also have a schedule associated with it (i.e. a course which may be given for two hours a day on Wednesdays and Thursdays).

To configure KonaKart for selling bookable products, the following steps must be taken :

- Using the Admin App, ensure that the "Product Type" attribute of all bookable products is set to "Bookable Product". When this is set, a sub folder appears which allows you to enter the specific data for bookable products:
 - The start and end dates are required attributes. They may be equal if a product is only valid for one day. Note that the Admin App automatically sets the time of the start date to be 00:00 and that of the end date to 23:59. If you are using the API directly, we recommend that you use the same convention. Once these have been entered, the "New Time Slot" button is enabled which allows you to insert time slots for the product. A time slot defines a start and end time for any day that lies between the start and end dates. A day may have multiple time slots. For example a course could be given for 2 hours in the morning and 1 hour in the afternoon.
 - Other attributes include the maximum number of bookings available and the current number of bookings, which is updated automatically as bookings are made. There are a number of custom fields that can be used to store information applicable to the type of bookable product.

If your application doesn't support bookable products, you may remove the "Bookable Product" entry from the product type drop lists in the Admin App, by setting the *Hidden Product Types* configuration variable from the *Configuration >> Admin App Configuration* section of the Admin App. As can be read in the Online Help, the id for Bookable Products is 6 .

Whenever a bookable product is purchased, the code that processes the order may be found in the OrderIntegrationMgr and the AdminOrderIntegrationMgr in the manageBookings() method. For performance reasons the code that calls this method (within the managers) is commented out and must be uncommented. The manageBookings() method parses the order and creates one or more booking objects which are saved in the database. These booking objects are related to the bookable product, the customer and to the order so that the bookings for any product, customer or order may be fetched using the KonaKart APIs.

The Admin App allows you to view and edit bookings. The Products Panel, Customers Panel and Orders Panel all have a "Bookings" button that opens up a Bookings panel that displays the bookings for the selected object. The button is not displayed if the role of the admin user doesn't allow him to view the Bookings panel.

The Application API contains a method (`getBookableProductConflict()`) that determines whether for a customer, a new booking may conflict with bookings that the customer has already made.

Import/Export of KonaKart Data

Export of Orders using `exportOrder`

An `exportOrder` API call is available for exporting a specified Order to a file.

The `exportOrder` interface requires an options parameters that is used to define the type of export to execute. Two examples that can be used for this code are:

- `KKConstants.EXP_ORDER_FULL_ORDER_TO_XML`

- KKConstants.EXP_ORDER_BY_SHIPPING_MODULE

When the KKConstants.EXP_ORDER_BY_SHIPPING_MODULE code is used, the idea is that the shipping module should provide an implementation of the order export that is specifically designed for integration with a shipping application for that shipping gateway.

An example of such a shipping module exportOrder implementation is provided for UPS for integration with UPS WorldShip. In this case the KonaKart order is exported as an XML file suitable for use with the Auto-Import feature of UPS WorldShip. The operator of UPS WorldShip should configure the Auto-Import directory (in UPS WorldShip) to be the directory where KonaKart exports its orders (by default, for store1 and UPS, this is under the KonaKart home directory in: orders/store1/2/ups/ where "2" is the value of the EXP_ORDER_BY_SHIPPING_MODULE constant, "ups" is the shipping module code, and "store1" is the storeId).

Two export order functions are available on the Orders panel of the Administration Application. These two export order functions correspond to the two export codes listed above (EXP_ORDER_FULL_ORDER_TO_XML and EXP_ORDER_BY_SHIPPING_MODULE). These functions are useful for ad-hoc order exports.

In some implementations of KonaKart it may be useful to export orders during normal order processing (rather than relying on manual intervention for this). There are many ways to implement the automatic export of orders. A common technique would be to use the OrderIntegrationMgr and AdminOrderIntegrationMgr to export orders whenever the order's status reached a certain value (such as "payment_received"). An example of doing just this is provided in the OrderIntegrtationMgr source code and all that is required to get it to work is to uncomment the calls.

Import/Export of KonaKart Data using XML_IO

XML_IO is a tool, provided in the Enterprise Extensions of KonaKart, that allows you to import and export KonaKart data in XML files.

Note the Warning section below. XML_IO does import compliant XML data files but in a specific way which may not be appropriate for your needs. Since KonaKart has a complete set of Administration APIs most users find it more flexible to write their own import/output routines (using the KonaKart KKAdminIf APIs) designed for their own specific-purposes than using the generic XML_IO facility.

It can be found in the xml_io directory which is located directly under the KonaKart installation directory.

The XML_IO utility can be configured to backup KonaKart systems, transfer product data from one system to another (e.g. from some kind of staging system to the live system), or any other use you can think of involving the import and export of XML data!

For some users it may be a useful way to export products or orders in XML format for subsequent communication with other systems. Indeed it could also be a useful way to import customers or products into KonaKart that are defined in other systems that can produce compatible XML files.

These are merely examples and it is expected that users will end up using the XML_IO utility in many different ways to suit their own specific requirements.

Configuration of XML_IO

A properties file allows you to define which KonaKart data objects you would like to be imported or exported providing tremendous flexibility of usage.

The properties file allows you to define the following:

When not modified, the default setting is true.

```
#accessoryProducts      = false
#addressFormats         = false
#audit                  = false
#bundledProducts        = false
#categories             = false
#categoriesToTagGroups = false
#configurationGroups   = false
#configurations         = false
#countries              = false
#coupons                = false
#crossSellProducts      = false
#customerGroups          = false
#customerTags            = false
#customerTagsForCusts   = false
#currencies             = false
#customers               = false
#dependentProducts      = false
#digitalDownloads       = false
#expressions             = false
#geoZones               = false
#ipnHistory              = false
#languages               = false
#manufacturers           = false
#productOptions          = false
#orders                  = false
#orderStatuses           = false
#productCategories       = false
#productOptionValues     = false
#products                = false
#productsToStores        = false
#reviews                 = false
#subZones                = false
#tagGroups               = false
#tagGroupToTags          = false
#tags                    = false
#taxClasses              = false
#taxRates                = false
#upSellProducts          = false
#wishLists               = false
#zones                   = false
```

The properties file allows you to define the data to be imported/exported using the Xml_IO import/export utility in KonaKart. You can call the properties file anything you like as its name is specified as an argument to the XML_IO command line utility (for details of the "usage" of the command line utility, see below).

If no properties are defined (or a properties file is not specified to the XML_IO utility) the default import/export configuration is used. By default all values are set to true. This means that by default all data is imported/exported. ** Note that this may take a long time if you have a lot of data!

If you define properties in this configuration properties file the values in it will override the default settings of the XML_IO utility.

A sample file is provided in the download kit for you to modify for your own purposes. Uncomment only the fields you want to define for import/export.

If you leave them commented out "true" is used by default.

If you are planning to import your exported data into another system where the Ids of the data are different, you should ensure that you export the appropriate dependent data objects (see below for some warnings about mapping of Ids). If, on the other hand, you are planning to import the exported data into a system where you know the Ids of the referenced objects will be the same, you can improve the performance of the export/import process by choosing only the higher level objects for export and import. If in doubt, you should export and import all the data objects so that you know you have a complete set of data for the site.

XML_IO Usage

You can discover the parameters for the XML_IO command line utility by issuing a "-?" as your argument as follows (example from a Linux system):

```
summersb@luton:~/konakart/xml_io$ ./setClasspath.sh
summersb@luton:~/konakart/xml_io$ ${JAVA_HOME}/bin/java -cp ${IMP_EXP_CLASSPATH} \
com.konakart.importer.xml.Xml_io -?

Usage: Xml_io
      (-i|-o)          Import or Export
      [-b bootstrapFile] Bootstrap file - an sql file run prior to importing
                           data to initialise the DB. If not set no bootstrap
                           is executed.
      [-e emptyDBFile]   An sql file used that's run prior to importing data
                           to empty the DB. If not set no empty DB is executed.
      [-r xmlRootDir]   Root directory of the XML data files on disk.
                           Default: ./xml_io
      [-d]              The directory is created if it doesn't exist.
      [-prop propsFile] Konakart Admin properties file name which must be
                           on the classpath. Default:
                           konakartadmin.properties
      [-conf confFile]  Configuration file that defines which parts of the
                           KonaKart database are imported/exported.
      [-usr userName]   User name to access the Konakart Admin engine.
                           Default: admin@konakart.com
      [-pwd password]  Password to access the Konakart Admin engine.
                           Default: princess
      [-s storeId]      The store to import to or export from.
                           Default: store1
      [-m engineMode]  KonaKart Engine Mode
                           0 = Single Store (default)
                           1 = Multi Store MultipleDB
                           2 = Multi Store SingleDB
      [-c (true|false)] Shared customers (only relevant in Engine Mode 2).
                           Default is false.
      [-ps (true|false)] Shared products (only relevant in Engine Mode 2).
                           Default is false.
      [-soap]           Use SOAP to access the Admin Engine.
                           Default is to use direct java calls
      [-ws url]         The endpoint for the Admin Web Services. Default:
                           http://localhost:8780/konakartadmin/services/KKWSAdmin
                           Only relevant when -soap is specified
      [-d]              to enable debug
      [-h|-?]          to show usage information
```

Warning

A *word of warning* when using the XML_IO utility! Loading data from one system to another can be a very complicated process so proceed with extreme caution. It is suggested that you backup your systems before executing XML_IO imports in the event that an import doesn't function the way you expect.

Complications arise in a number of different areas. One is in the case where exported Ids (and keys) from one system do not match those on the target system to import into. The XML_IO utility does have the "intelligence" necessary to "map" these ids during the import process and does this in a number of different ways. One way to ensure the mappings are correct is to include the necessary reference data in the XML_IO export so that this can be used during the import process by XML_IO to figure out the correct mappings. An example is to include the languages export when you export the products so that when the products are subsequently loaded into a different system (where languages could easily be defined with different Ids) the language Ids can be mapped on the imported products.

Nevertheless, it is important to note that even when you include all the necessary reference data there are, unavoidably, occasions when XML_IO cannot calculate a mapping successfully. This can occur, for example, when it tries to locate a language by name in the target system (in order to figure out the mapping

of Ids) but fails to find exactly one match. In such cases the XML_IO utility will report the mapping failure as a "Warning" in the log.

Therefore, it is strongly advised that you check your import/export processes carefully before unleashing in production. XML_IO is a very powerful utility but if used incorrectly can lead to duplicate records and in some cases a loss of integrity in your database.

KonaCart has a complete set of Administration APIs so most users find it more flexible to write their own import/output routines designed for their own specific-purposes than using the generic XML_IO facility.

Examples

In the xml_io directory, located under the KonaCart installation home directory, you will find some example scripts for running XML_IO either directly using the "engine" or via SOAP. These may be useful as they stand or they could be used as templates for your own XML_IO scripts that are tailored for your own particular environment.

Custom Imports Using the Importer Panel

The Enterprise version of the Administration Application provides an Importer Panel that allows you to define your own custom Import routines that you can assign to custom import buttons.

There are three custom import buttons available for three custom commands. An example import routine is provided and set up on custom import button number 1. This example imports products from an XML file (that you select before clicking on Import) whose format definition can be found under the installation home in "\xml_io\schema\konakart_xml_io.n.n.n.n.xsd" (refer to AdminProductsXML).

Note that this particular import is just a simple example. It only imports products with SKUs that are not specified or do not already exist in the KonaCart database. It does not update products at all. It does not attempt to match the names of referenced objects but just uses the Ids specified in the data file. If, for example, a product's manufacturer Id refers to a manufacturer that doesn't exist, the product insert will fail.

The example is provided just to give you a starting point for your own custom import routines and show you how to make them accessible to your Admin users. It is likely that you will have specific requirements for your imports that will not be satisfied by this simple example.

The example source code can be found under the installation home at "xml_io/src/com/konakart/importer/xml/ImportXml.java"

Configuration of the Importer Panel

Using File-based configuration (in konakartadmin_gwt.properties) you can define any custom import routine to be associated with each of the three import buttons.

For example, by default, the example import routine is set up on custom import button 1:

```

#-----
#
# Importer Configuration
#
#-----
#
# Allows you to specify custom jobs on each of the 3 Custom Import
# buttons.
#
# Enterprise Only
#
# Set the custom import button labels in the AdminMessages.properties file
# for each of your supported languages in these properties:
#
# button.importer.custom1
# button.importer.custom2
# button.importer.custom3
#
# If the job class is undefined for a Custom Import button
# it will not be shown on the Importer Panel.
#-----

fbc.import.custom1.job.class = com.konakart.importer.xml.ImportXml
fbc.import.custom1.job.method = importXml

#fbc.import.custom2.job.class = custom2.class
#fbc.import.custom2.job.method = custom2.method

#fbc.import.custom3.job.class = custom3.class
#fbc.import.custom3.job.method = custom3.method

```

Your custom routine must have a constructor that takes a KKAdminIf object and the method that you specify must receive an array of Strings. For example:

```

/**
 * Constructor that takes an Admin Engine
 *
 * @param adEng
 *          KKAdminIf engine
 */
public ImportXml(KKAdminIf adEng)
{
    // You will probably want to save the engine:
    setAdminEng(adEng);
}

/**
 * ImportXml using the specified command line args
 *
 * @param args arguments: will be called by the Admin App with:
 *             -se sessionId (of the logged-in admin user)
 *             -i comma-separated list of filenames or directories
 *             -o log-filename
 *
 * The log filename will use a concatenation of the selected files
 * with a file-friendly timestamp and ".log" appended.
 */
public void importXml(String[] args)
{
    // Add your custom import code !
}

```

As for all button labels, the custom import button labels are defined in the AdminMessages.properties file for each supported language. The default (English) version defines these:

```
button.importer.custom1      = Import
button.importer.custom2      = Import2
button.importer.custom3      = Import3
```

By default the Importer Panel appears as follows: (showing the online help as well):

Welcome back, admin@konakart.com
[Set Language](#) [Change Password](#) [Sign Out](#) [About](#)

Importer

productsExample.xml

Help

KonaKart Importer

This panel allows you to import data from files into KonaKart using custom import routines. Please refer to the User Guide for more details.

For the **Import** and **Delete** commands you first need to select one or more files to operate on before clicking on the respective buttons.

By default, the **Import** button is configured to load products into KonaKart from the selected file. An example datafile (productsExample.xml) is provided which is an example of the format of the file that is required for this simplified import routine.

You can upload new data files for subsequent import by clicking on the **Upload** button.

The **Refresh** button updates the set of files in the directory specified to hold the data files. You can configure the location of the data files directory on the panel at **Configuration >> Importer Configuration**. Likewise you can configure where the importer log files will be written on the same configuration panel.

Close

Importer Panel - also showing the on-line help

Importer Log Files

Log files created by the importer jobs can be viewed from the "View Importer Logs" panel. The location for the importer log files can be specified in a configuration variable - See Configurations >> Importer Configurations.

Reset Database Tool

The Enterprise version of the Administration Application provides a Reset Database Tool that can be executed to reset your KonaKart database to a state containing just a minimal set of reference data (and customers) ready for you to add your own categories, manufacturers and products.

Therefore, it should be used with extreme care. Do not remove all your products by accident!

Configuration of the Reset Database Tool Option

You can replace the name of the command on the Tools menu with the name of a custom tool of your own specification and change the definition of the class and method that is executed to change what happens when that option is selected.

Using File-based configuration (in konakartadmin_gwt.properties) you can define any custom routine to be associated with the Reset Database option.

By default, if the values in the konakartadmin_gwt.properties are commented out, the emptyDatabase method of the com.konakartadmin.utils.ResetDatabase class is executed. You can specify your own values here instead to change the functionality:

```
#-----
# Tools Configuration
#
# -----
#
# Allows you to specify a custom tool to replace the Reset Database
# operation.
#
# Enterprise Only
#
# Specify your own class and method to repace the Reset Database
# operation with your own.
# Default values are shown commented out
#
#-----
#fbc.tools.resetdb.job.class = com.konakartadmin.utils.ResetDatabase
#fbc.tools.resetdb.job.method = emptyDatabase
```

Your custom routine must have a constructor that takes a KKAdminIf object and the method that you specify must receive an array of Strings. For example:

```
/**
 * Constructor that takes an Admin Engine
 *
 * @param adEng
 *          KKAdminIf engine
 */
public ResetDatabase(KKAdminIf adEng)
{
    // You will probably want to save the engine:
    setAdminEng(adEng);
}

/**
 * Empty Database using the specified command line args
 *
 * @param args arguments: will be called by the Admin App with:
 *             -se sessionId (of the logged-in admin user)
 *             -o output-filename
 *
 * The output filename will use the method name that you define
 * with ".log" appended so in the case of the default example,
 * this is emptyDatabase.log
 */
public void emptyDatabase(String[] args)
{
    // Add your custom code !
}
```

As for all Admin Application messages, the Reset database messages can be changed by modifying the following properties in the AdminMessages.properties file for each supported language. The default (English) version defines these:

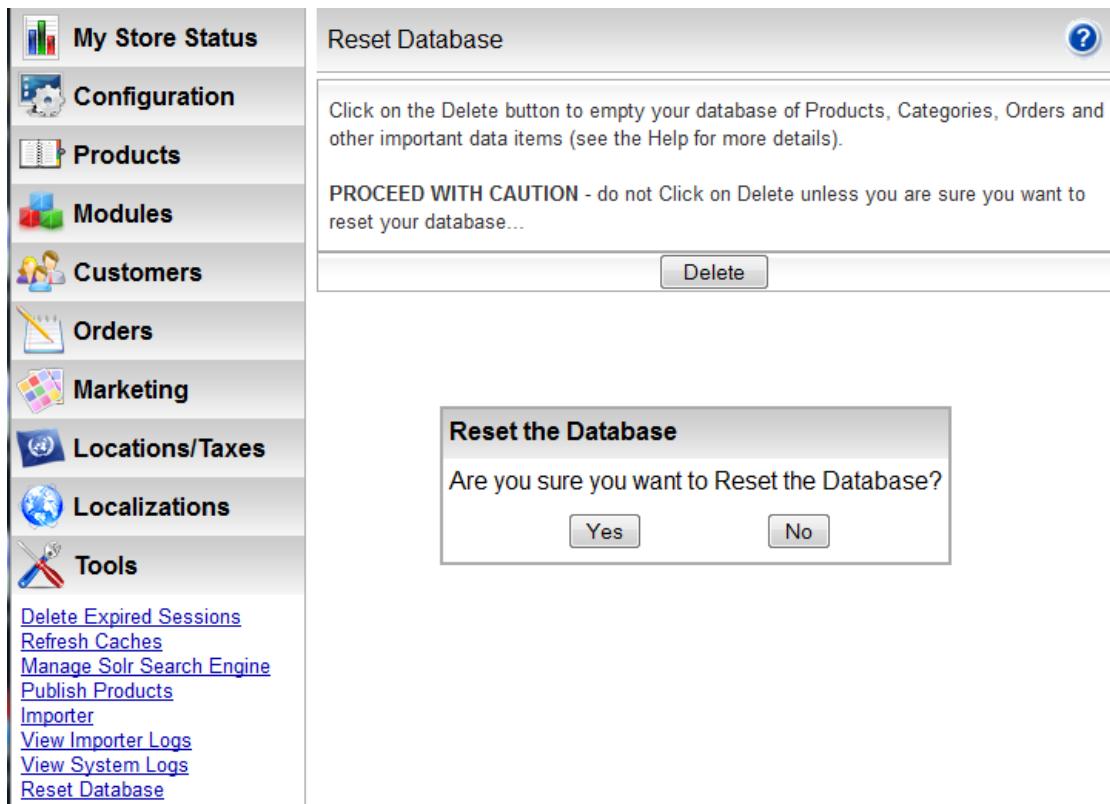
```

panel.resetDatabase      = Reset Database
question.resetDatabase   = Are you sure you want to Reset the Database?
msg.resetDatabase        = Click on the Delete button to empty your database of
                           Products\, Categories\, Orders and other important data
                           items (see the Help for more details).<br /><br /><b>PROCEED
                           WITH CAUTION</b> - do not Click on Delete unless you are
                           sure you want to reset your database...
msg.resettingDatabase   = Resetting Database...
help.resetDatabase       = Reset the Database

```

You can also change the help text that is associated with the panel by changing the help.resetDatabase property in the various AdminHelpMessages.properties files.

By default the Reset Database option appears as follows: (after clicking on the Delete button you get a warning before proceeding):



Reset Database Panel - with warning dialog

View System Log Files

A Log file is created by the Reset Database command. This, as well as other system log files, can be viewed from the "View System Logs" panel.

Multiple Prices for Products

Each KonaKart product object has 4 price fields that can be used to store 4 different prices. For example, these may be retail price, wholesale price, MRP, employee price etc.

The prices may be stored just for reporting purposes since whenever a product is added to an Order and an OrderProduct object is created which becomes an attribute of the Order object, this OrderProduct contains all of the product prices. This means that whenever documentation is generated from the order, you can show customers information such as the discount from the MRP etc.

The prices may also be used to display different prices to different customers based on what customer group they belong to.

In order to have an unlimited number of prices, they may be stored in a separate database table and referenced by a catalog id. The database table is called kk_product_prices. In order to pick up the external prices, the Application Engine must be configured with the correct catalog id. The catalog id can be calculated in a number of ways and is application specific. For example, it may depend on the country that the customer is accessing from, or the URL or the type of customer etc. One way of configuring the application engine is in the KKAppEngCallouts class as shown below:

```
/**
 * Callouts to add custom code to the KKAppEng
 */
public class KKAppEngCallouts
{
    /**
     * Called at the start of startup
     *
     * @param eng
     */
    public void beforeStartup(KKAppEng eng)
    {
        System.out.println("Set product options for current customer");
        FetchProductOptionsIf fpo = new FetchProductOptions();
        fpo.setCatalogId("cat1");
        fpo.setUseExternalPrice(true);
        fpo.setUseExternalQuantity(true);
        eng.setFetchProdOptions(fpo);
    }

    /**
     * Called at the end of startup
     *
     * @param eng
     */
    public void afterStartup(KKAppEng eng)
    {
    }
}
```

The catalogId must be set to the id of a valid catalog and the the UseExternalPrice attribute must be set to true. Note that when using the application API directly, this functionality is available on all of the API calls that accept an Options object such as FetchProductOptions or AddToBasketOptions. The Options object for each API call must be configured in a similar way as shown above.

The kk_product_prices table may be loaded directly using the appropriate database tools or it may be loaded via the Admin App API using the insertProductWithOptions() or editProductWithOptions() API calls. The AdminProductMgrOptions object must be set with the correct catalogId and configured to use external prices. When the call is made, the product and product attribute prices are written to the kk_product_prices table rather than to the standard product tables.

The Admin App may also be used to maintain the external product prices. The Catalog must first be defined using the Catalogs panel. Once defined, it appears in a drop list in the panel header. When selected in the drop list, all product related API calls made by the admin app attempt to use the catalog. i.e. If you edit a product, the prices, attribute prices and tier prices are all written to the kk_product_prices table rather than to the standard tables. When inserting a product, the prices are written both to the kk_product_prices table as well as the standard tables.

Note that if a catalog is defined but the product does not have entries in the kk_product_prices table, then it will not be displayed in the Products Panel after a search. In order to add the prices to the kk_product_prices table you must take the following steps:

- Select "No Catalog" from the catalog drop list.
- Click the Edit button for the selected product.
- Modify the prices for that product.
- Select the catalog from the drop list.
- Click the Save button to save the prices in the kk_product_prices table for the chosen catalog.

Note that the API calls using the kk_product_prices table are only available when the Enterprise Extensions are installed.

Sale Price

A sale price may be defined for each product with a start and end date for the sale period. The price is defined using the Admin App, in the Prices tab of the Edit Product panel.

Special Price					
Special Price:	22.99				
Enable Special:	<input checked="" type="checkbox"/>				
Start Date:	21/07/2010	Hours:	0	Minutes:	0
Expiry Date:	23/07/2010	Hours:	23	Minutes:	59

The image above shows how the special price is defined for a product. Below, we show an example of how the sale price may be displayed in the storefront application by using a strike-through-

Frantic ~~\$35.00~~ **\$22.99**
[DVD-FRAN]

In the Enterprise version of KonaCart, a batch job is supplied, called AdminProductBatchMgr.setSpecialPriceStateBatch() which disables the special price if it has expired or isn't active yet. The reason for disabling the special price is that if not disabled, the special price is still used to order and filter products by price even though the current date may not lie within the defined date range (This is done for performance reasons).

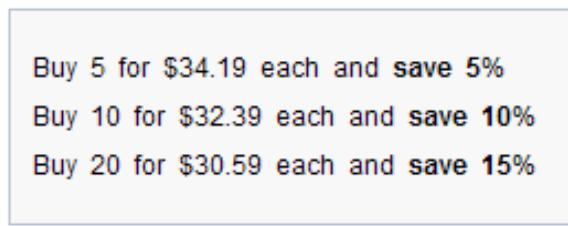
Note that special price functionality is not supported when using catalog prices.

Tier Pricing

Tier Pricing is a way to encourage shoppers to buy larger quantities of a product by applying discounts based on the quantity ordered. These discounts may be "tiered" so that they increase as the order amount is raised. KonaKart supports the definition of actual tier prices as well as percentage discounts. The prices are defined using the Admin App, in the Prices tab of the Edit Product panel.

Tier Prices																																																																							
New Tier Price																																																																							
<table border="1"> <tr> <td>Quantity <input type="text" value="5"/></td> <td>and above</td> <td>Price: <input type="text" value="34.19"/></td> <td>Price1: <input type="text"/></td> <td>Price2: <input type="text"/></td> <td>Price3: <input type="text"/></td> </tr> <tr> <td colspan="6" style="text-align: right;">Delete</td> </tr> <tr> <td colspan="6"> <input type="checkbox"/> Use Percentage Discount: </td> </tr> <tr> <td colspan="6"> <table border="1"> <tr> <td>Quantity <input type="text" value="10"/></td> <td>and above</td> <td>Price: <input type="text" value="32.39"/></td> <td>Price1: <input type="text"/></td> <td>Price2: <input type="text"/></td> <td>Price3: <input type="text"/></td> </tr> <tr> <td colspan="6" style="text-align: right;">Delete</td> </tr> <tr> <td colspan="6"> <input type="checkbox"/> Use Percentage Discount: </td> </tr> <tr> <td colspan="6"> <table border="1"> <tr> <td>Quantity <input type="text" value="20"/></td> <td>and above</td> <td>Price: <input type="text" value="30.59"/></td> <td>Price1: <input type="text"/></td> <td>Price2: <input type="text"/></td> <td>Price3: <input type="text"/></td> </tr> <tr> <td colspan="6" style="text-align: right;">Delete</td> </tr> <tr> <td colspan="6"> <input type="checkbox"/> Use Percentage Discount: </td> </tr> </table> </td> </tr> </table></td></tr></table>						Quantity <input type="text" value="5"/>	and above	Price: <input type="text" value="34.19"/>	Price1: <input type="text"/>	Price2: <input type="text"/>	Price3: <input type="text"/>	Delete						<input type="checkbox"/> Use Percentage Discount:						<table border="1"> <tr> <td>Quantity <input type="text" value="10"/></td> <td>and above</td> <td>Price: <input type="text" value="32.39"/></td> <td>Price1: <input type="text"/></td> <td>Price2: <input type="text"/></td> <td>Price3: <input type="text"/></td> </tr> <tr> <td colspan="6" style="text-align: right;">Delete</td> </tr> <tr> <td colspan="6"> <input type="checkbox"/> Use Percentage Discount: </td> </tr> <tr> <td colspan="6"> <table border="1"> <tr> <td>Quantity <input type="text" value="20"/></td> <td>and above</td> <td>Price: <input type="text" value="30.59"/></td> <td>Price1: <input type="text"/></td> <td>Price2: <input type="text"/></td> <td>Price3: <input type="text"/></td> </tr> <tr> <td colspan="6" style="text-align: right;">Delete</td> </tr> <tr> <td colspan="6"> <input type="checkbox"/> Use Percentage Discount: </td> </tr> </table> </td> </tr> </table>						Quantity <input type="text" value="10"/>	and above	Price: <input type="text" value="32.39"/>	Price1: <input type="text"/>	Price2: <input type="text"/>	Price3: <input type="text"/>	Delete						<input type="checkbox"/> Use Percentage Discount:						<table border="1"> <tr> <td>Quantity <input type="text" value="20"/></td> <td>and above</td> <td>Price: <input type="text" value="30.59"/></td> <td>Price1: <input type="text"/></td> <td>Price2: <input type="text"/></td> <td>Price3: <input type="text"/></td> </tr> <tr> <td colspan="6" style="text-align: right;">Delete</td> </tr> <tr> <td colspan="6"> <input type="checkbox"/> Use Percentage Discount: </td> </tr> </table>						Quantity <input type="text" value="20"/>	and above	Price: <input type="text" value="30.59"/>	Price1: <input type="text"/>	Price2: <input type="text"/>	Price3: <input type="text"/>	Delete						<input type="checkbox"/> Use Percentage Discount:					
Quantity <input type="text" value="5"/>	and above	Price: <input type="text" value="34.19"/>	Price1: <input type="text"/>	Price2: <input type="text"/>	Price3: <input type="text"/>																																																																		
Delete																																																																							
<input type="checkbox"/> Use Percentage Discount:																																																																							
<table border="1"> <tr> <td>Quantity <input type="text" value="10"/></td> <td>and above</td> <td>Price: <input type="text" value="32.39"/></td> <td>Price1: <input type="text"/></td> <td>Price2: <input type="text"/></td> <td>Price3: <input type="text"/></td> </tr> <tr> <td colspan="6" style="text-align: right;">Delete</td> </tr> <tr> <td colspan="6"> <input type="checkbox"/> Use Percentage Discount: </td> </tr> <tr> <td colspan="6"> <table border="1"> <tr> <td>Quantity <input type="text" value="20"/></td> <td>and above</td> <td>Price: <input type="text" value="30.59"/></td> <td>Price1: <input type="text"/></td> <td>Price2: <input type="text"/></td> <td>Price3: <input type="text"/></td> </tr> <tr> <td colspan="6" style="text-align: right;">Delete</td> </tr> <tr> <td colspan="6"> <input type="checkbox"/> Use Percentage Discount: </td> </tr> </table> </td> </tr> </table>						Quantity <input type="text" value="10"/>	and above	Price: <input type="text" value="32.39"/>	Price1: <input type="text"/>	Price2: <input type="text"/>	Price3: <input type="text"/>	Delete						<input type="checkbox"/> Use Percentage Discount:						<table border="1"> <tr> <td>Quantity <input type="text" value="20"/></td> <td>and above</td> <td>Price: <input type="text" value="30.59"/></td> <td>Price1: <input type="text"/></td> <td>Price2: <input type="text"/></td> <td>Price3: <input type="text"/></td> </tr> <tr> <td colspan="6" style="text-align: right;">Delete</td> </tr> <tr> <td colspan="6"> <input type="checkbox"/> Use Percentage Discount: </td> </tr> </table>						Quantity <input type="text" value="20"/>	and above	Price: <input type="text" value="30.59"/>	Price1: <input type="text"/>	Price2: <input type="text"/>	Price3: <input type="text"/>	Delete						<input type="checkbox"/> Use Percentage Discount:																													
Quantity <input type="text" value="10"/>	and above	Price: <input type="text" value="32.39"/>	Price1: <input type="text"/>	Price2: <input type="text"/>	Price3: <input type="text"/>																																																																		
Delete																																																																							
<input type="checkbox"/> Use Percentage Discount:																																																																							
<table border="1"> <tr> <td>Quantity <input type="text" value="20"/></td> <td>and above</td> <td>Price: <input type="text" value="30.59"/></td> <td>Price1: <input type="text"/></td> <td>Price2: <input type="text"/></td> <td>Price3: <input type="text"/></td> </tr> <tr> <td colspan="6" style="text-align: right;">Delete</td> </tr> <tr> <td colspan="6"> <input type="checkbox"/> Use Percentage Discount: </td> </tr> </table>						Quantity <input type="text" value="20"/>	and above	Price: <input type="text" value="30.59"/>	Price1: <input type="text"/>	Price2: <input type="text"/>	Price3: <input type="text"/>	Delete						<input type="checkbox"/> Use Percentage Discount:																																																					
Quantity <input type="text" value="20"/>	and above	Price: <input type="text" value="30.59"/>	Price1: <input type="text"/>	Price2: <input type="text"/>	Price3: <input type="text"/>																																																																		
Delete																																																																							
<input type="checkbox"/> Use Percentage Discount:																																																																							

The image above shows how tier prices may be defined for a product. Below, we show an example of how these prices may be displayed in the storefront application.



The image below displays an example of how percentage based tier prices may be defined. The percentage amount is entered in the normal price field and the "Use Percentage Discount" checkbox is checked.

Tier Prices					
New Tier Price					
Quantity <input type="text" value="5"/>	and above	Price: <input type="text" value="6.00"/>	Price1: <input type="text"/>	Price2: <input type="text"/>	Price3: <input type="text"/>
		Cust1: <input type="text"/>	Delete		
Use Percentage Discount: <input checked="" type="checkbox"/>					
Quantity <input type="text" value="10"/>	and above	Price: <input type="text" value="8.00"/>	Price1: <input type="text"/>	Price2: <input type="text"/>	Price3: <input type="text"/>
		Cust1: <input type="text"/>	Delete		
Use Percentage Discount: <input checked="" type="checkbox"/>					
Quantity <input type="text" value="15"/>	and above	Price: <input type="text" value="10.00"/>	Price1: <input type="text"/>	Price2: <input type="text"/>	Price3: <input type="text"/>
		Cust1: <input type="text"/>	Delete		
Use Percentage Discount: <input checked="" type="checkbox"/>					
Quantity <input type="text" value="20"/>	and above	Price: <input type="text" value="12.00"/>	Price1: <input type="text"/>	Price2: <input type="text"/>	Price3: <input type="text"/>
		Cust1: <input type="text"/>	Delete		
Use Percentage Discount: <input checked="" type="checkbox"/>					

Below, we show an example of how percentage based tier pricing may be displayed in the storefront application.

Buy 5 and save 6%
Buy 10 and save 8%
Buy 15 and save 10%
Buy 20 and save 12%

Dynamic product prices

The price of a product may be defined at the moment that it is added to the cart. This functionality is useful when the application logic is responsible for calculating the product price.

```
BasketIf b = new Basket();
b.setQuantity(1);
b.setProductId(selectedProd.getId());
b.setFinalPriceExTax(new BigDecimal("22.22"));
kkAppEng.getBasketMgr().addToBasket(b, /* refresh */true);
```

In order to define your own price, you must set the FinalPriceExTax attribute of the Basket object as shown above. Normally this isn't set and the price is retrieved from the Product. The above example is taken from one of the storefront application action classes which is communicating with the KonaKart client engine. The concept is the same when communicating directly with the server engine.

In order to allow this functionality, the "Allow to use Basket Price" configuration variable must be set. This can be done through the Admin App under Configuration >> Security and Auditing. Note that if the Web Services interface is publicly available, a programmer has the possibility of writing a utility to add a product to the cart with any price. Therefore we advise using this functionality only when the Web Services interface isn't available over the web (the normal case).

Tax Configuration

KonaKart includes a powerful tax calculation system containing the following entities:

- Zone: A physical area and is normally a state or region within a country.
- Tax Area: An area defined for tax purposes and can be mapped to a zone or a country.
- Tax Class: Defines a type of product for tax purposes because products within the same tax area may be taxed differently depending on their tax class. For example, children's clothes or basic necessities such as milk, may be taxed differently to luxury goods.
- Tax Rate: A percentage number used to actually calculate the tax on a product. It has to be associated with a tax area and a tax class.

To start, you should define all of the zones for the country that you are interested in. Our database comes pre-populated with zones for a few countries. Even if the tax rate doesn't change between zones, it is still a good idea to populate them since this helps customers when entering addresses because they can be displayed in a drop down list.

Next, you should define the tax areas which may map directly to the zones. For example, the state of Texas may map directly to a tax area. Once you've defined all of your tax areas, you should map them to the zones using the Admin App.

The following step is to define the tax classes which are needed before entering any products. Once these are defined, the final step is to insert all of the tax rates that could be applied and map them with a tax area and a tax class.

If for some reason you need to look up the tax rate to be applied for a product from an external system, this rate can be applied to the order before saving it. In order to do this you must loop through all of the OrderProduct objects contained in the order and call the setTaxRate() method to set the tax rate. Once

this has been done, you can call the `getOrderTotals()` engine call which re-calculates the order for the new tax rates.

Multiple Tax Areas Within a State

In the US there are cases where a state contains multiple tax areas which may be determined by the zip code or a combination of zip code and county or some other details of the address. In this case, all of these physical zones may be added to the list of zones but made invisible so that when displaying the list of zones for a country, the list box does not show thousands of zones. For example, lets say that state ABC has 3000 different tax areas delimited by zip code. In this case all of these zones could be stored by KonaKart and all display the same name (i.e. state ABC). Therefore the zones table will contain 3001 zones for state ABC, 3000 of which are invisible and so are not returned by the `getZonesPerCountry()` API call. When registering, the customer selects the one visible zone called ABC and clicks the register button. Before calling the `RegisterCustomer()` API call, some custom code needs to call the `searchForZones()` API call with the zip code and state code as search criteria in order to get the correct zone. Once the correct zone has been fetched, the custom code must set the zone id in the `CustomerRegistration` object so that the registration uses that id. Note that the `Zone` object has a search field which must be set when inserting the zone to text that can be used to search for it (i.e. zip code in this example).

The procedure for handling the case when a customer is editing his address, is slightly different. In this case the customer may have registered with a `zoneId` that points to an invisible zone and so the zone is not present when a list is returned from the `getZonesPerCountry()` API call. Therefore, there needs to be custom code that retrieves the customer's zone using the `searchForZones()` API call and that removes the generic zone that describes the state so that the state does not appear twice. As in the `RegisterCustomer()` case, when the customer clicks the save button, the custom code must figure out the real zone of the customer using data within the address and populate the `zoneId` attribute of the address object before saving. The `useZoneId` attribute of the address object must also be set to true to stop the server code from attempting to determine the customer's zone.

Tax algorithm and numeric precision

Different countries and tax jurisdictions tend to define different ways of calculating tax. There are various ways of customizing KonaKart in order to make it compliant.

The configuration variable labeled "No of decimal places for currency" in the Configuration >> Admin App Configuration section of the Admin App allows you to set the precision of the price entry fields. Although your currency only uses two decimal places, you may want to enter net prices to more decimal places. If you are entering net prices but displaying prices after tax has been added (gross prices) you may need this extra precision in order to enable you to generate all gross prices. This can be set differently for each store in a multi-store environment. The precision defined by this variable is also the precision used for tax calculations.

The configuration variable labeled "Tax Quantity Rule" in the Configuration >> Stock and Orders section of the Admin App allows you to set the algorithm used for making the tax calculation. The actual code that calculates the tax is supplied in source code format under `KonaKart/custom/utils/src/com/konakart/util` and is called `TaxUtils.java`. As you can see from the code within `TaxUtils.java` there are currently two algorithms defined by the variables:

- `public static final int TAX_ON_TOTAL = 1`
- `public static final int TAX_PER_ITEM = 2`

The default `TAX_ON_TOTAL` algorithm calculates the total cost of an order line item by multiplying the unit price with the quantity and then calculates the tax. The `TAX_PER_ITEM` algorithm calculates

and rounds the tax for a single item before multiplying the result by the quantity. The latter is useful in countries where the prices are displayed inclusive of tax and are calculated from the net price. Using this algorithm you are assured that the displayed price of an article remains consistent when the quantity bought is greater than 1.

If the current implementations defined within TaxUtils.java do not meet your requirements, you may customize this class and compile the modified code using the techniques defined elsewhere within this document.

Validation of Order Totals

KonaKart supports tax and shipping modules that interface to external services in order to gather the requested data. These services may not always be available or may not return relevant data because of erroneous input data such as a non existent address.

One way of circumventing the problem for shipping modules is to provide a backup table driven module that only returns a quote if the primary module is unable to. This method always guarantees a shipping quote, allowing your customer to complete the transaction and to checkout. The table driven module may not return a 100% accurate shipping cost, but many merchants are happy to take a small risk on shipping charges rather than to risk an abandoned cart.

An alternative approach is to not allow your customer to checkout if all of the required order total modules do not exist. KonaKart includes code that allows you validate the order total modules using your own business rules. The code is in the *KKAppEngCallouts* class in a method called *public boolean validateOrderTotals(KKAppEng eng, OrderIf order)* . This class may be found under the *custom\appn\src\com\konakart\al* directory. The method provides sample code that determines whether certain modules are active and if so, checks to see whether an order total exists on the order for the active module. If it doesn't exist, the result is passed back to the customer in the form of a popup panel in the order confirmation screen, informing him that he cannot checkout because a problem has been encountered.

Multiple Quantities for Products

Each KonaKart product object has a quantity field that can be used to store the quantity of the product in stock. There is also an availability date used to store the date that the product will become available if it is out of stock. Configurable products (i.e. red shirt, size medium) have separate quantity and available date attributes for each configuration.

As is the case for product prices, a product may be associated with multiple quantities and available dates. This quantity data is stored in a separate database table called *kk_catalog_quantity* and the key that determines which quantity data is actually used is the *catalogId*.

One way of configuring the application engine to use external quantity data is in the *KKAppEngCallouts* class as shown below:

```

/**
 * Callouts to add custom code to the KKAppEng
 */
public class KKAppEngCallouts
{
    /**
     * Called at the start of startup
     *
     * @param eng
     */
    public void beforeStartup(KKAppEng eng)
    {
        System.out.println("Set product options for current customer");
        FetchProductOptionsIf fpo = new FetchProductOptions();
        fpo.setCatalogId("cat1");
        fpo.setUseExternalPrice(true);
        fpo.setUseExternalQuantity(true);
        eng.setFetchProdOptions(fpo);
    }

    /**
     * Called at the end of startup
     *
     * @param eng
     */
    public void afterStartup(KKAppEng eng)
    {
    }
}

```

The catalogId must be set to the id of a valid catalog and the the UseExternalQuantity attribute must be set to true. Note that when using the API directly, this functionality is available on all of the application API calls that accept an Options object such as FetchProductOptions or AddToBasketOptions. The Options object for each API call must be configured in a similar way as shown above.

The kk_catalog_quantity table may be loaded directly using the appropriate database tools or it may be loaded and read through the Admin App API using various API calls:

- setProductQuantityWithOptions()
- getProductQuantityWithOptions()
- setProductAvailabilityWithOptions()
- getProductAvailabilityWithOptions()
- getProductWithOptions()
- editProductWithOptions()

When any of the above calls are made with the options object configured to use external quantities, the quantity data is written to and read from the kk_catalog_quantity table rather than the standard product tables.

The Admin App may also be used to maintain the external product quantities. The Catalog must first be defined using the Catalogs panel. Once defined, it appears in a drop list in the panel header. When selected in the drop list, all product related API calls made by the admin app attempt to use the catalog. i.e. If you edit a product, the quantities are all written to the kk_catalog_quantity table rather than to the standard tables. When inserting a product, the quantities are written both to the kk_catalog_quantity table as well as the standard tables.

Note that if a catalog is defined but the product does not have entries in the kk_catalog_quantity table, then it will not be displayed in the Products Panel after a search. In order to add the quantities to the kk_catalog_quantity table you must take the following steps:

- Select "No Catalog" from the catalog drop list.
- Click the Edit button for the selected product.
- Modify the quantities for that product.
- Select the catalog from the drop list.
- Click the Save button to save the quantities in the kk_catalog_quantity table for the chosen catalog.

Note that the API calls using the kk_catalog_quantity table are only available when the Enterprise Extensions are installed. Also note that special price functionality is not supported when using catalog prices.

Default Sort Order for Products

To set the default sort order for products retrieved using the client API you must use the *setDefaultOrderBy()* call on the com.konakart.al.ProductMgr. This sets the default sort order used by the following calls:

- fetchProductsPerCategory()
- fetchAllSpecials()
- searchForProducts()
- fetchProductsPerManufacturer()
- fetchAlsoPurchasedArray()
- fetchRelatedProducts()

setDefaultOrderBy() should be called in the struts action class prior to making the API call that actually fetches the data from the DB. The parameter accepted by *setDefaultOrderBy()* must be an attribute of *com.konakart.app.DataDescConstants* such as ORDER_BY_NAME_ASCENDING or ORDER_BY_MANUFACTURER etc.

Bundle Configuration

From version 2.2.6.0, KonaKart supports bundled products. To define a bundle, the following steps must be taken using the Admin App:

- When inserting or editing a product, select a product type of "Bundle" or "Bundle with free shipping".
- Using the "Select Products" button which appears next to the Product Type drop list, open a pop-up window to select the bundled products.
- When you return to the main window after having selected the bundled products, you can get the Admin App to calculate the price and the weight of the bundle. When calculating the price, a percentage or total discount may be configured. The calculated values can be overridden before saving. The quantity is read only and is always a calculated value based on the quantity of bundled products in stock.

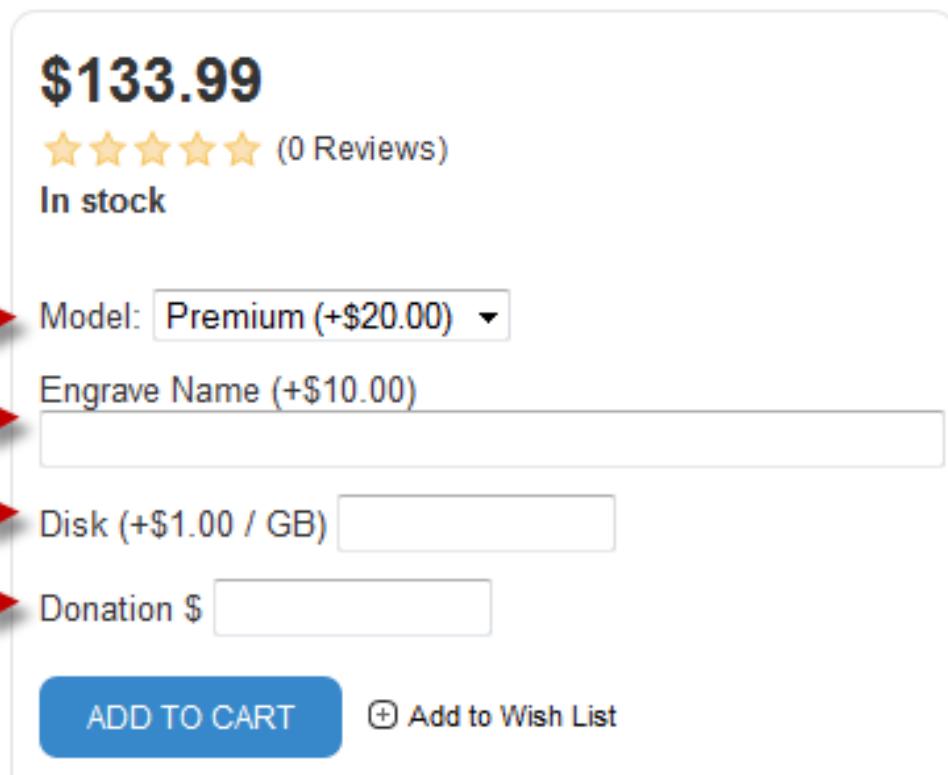
Within the application, bundle products are treated in the same way as other products. The products that make up the bundle (bundled products) can be retrieved through the API using the *getRelatedProducts()* API call. *ProductDetailsBody.jsp* has been modified to display description information for each bundled product when it exists. Just like other products, the quantity in stock of a bundle product is retrieved using

the updateBasketWithStockInfo() method. The quantity returned is calculated based on the availability of the bundled products and when a bundle product is sold, the stock info of the bundled products is modified.

Product Options

Product options allow a customer to configure a product before adding it to the cart. Options may be created and maintained using the Products >> Product Options panel in the Admin App. An option may be added to a product in the Attributes folder of the Edit Product panel. When adding an option to a product, a positive or negative price may be defined which will be added to or subtracted from the main product price when the option is added to the product.

KonaCart supports four different option types. The image below shows how they are rendered in the standard storefront application.



1. The Standard type is where each option value defines a product attribute which may or may not modify the price of the product (e.g. Small, Medium and Large for an option called Size or Black, Brown and Red for an option called Color). The above image shows an option called Model with a value called Premium that increases the price of the product by \$20.00.
2. The Customer Text type (Enterprise Only) option allows a customer to enter text which for example could be used to personalize the product with a name, initials or a greeting. In this case the option could be called Engrave and the option value called Name. The option may be given a price which will be added to the product price in order to charge for the customization service. The above image shows an increment of \$10.00 if text is added.
3. The Variable Quantity type (Enterprise Only) is an option, the value of which may take a customer defined quantity. For example if the option is named Disk you may define one option value called GB and in the storefront UI a data entry text box is displayed to the user (rather than a list of fixed option

values) where the user may enter the number of Gigabytes of additional disk space he requires. In this case the price defined when adding the option to a product, becomes the unit price and will be multiplied by the quantity entered by the customer in order to determine the final price of the option.

- The Customer Price type (Enterprise Only) allows the customer to enter a price for the option. This could for example be used for making donations where the customer can enter the donation amount. In this case the option could be called Donation and one option value could be defined named \$. In the storefront UI a data entry text box is displayed to the user where an amount may be entered. The option price is ignored and the price taken is the amount entered by the customer which is added to the product price. In the case of a donation the product price could be set to zero.

Product Tags

From version 2.2.6.0, KonaCart supports product tags. Tags are attributes that can be associated to a product and can be used to refine product searches. For example, in the case of a digital camera the tags could be arranged as follows:

MegaPixels	Optical Zoom	Weight
6.0	3x	less than 100g
8.0	4x	between 100g and 200g
10.0	5x	greater than 200g

The Optical Zoom tags (3x,4x and 5x) are contained within a tag group called Optical Zoom. All tags must belong to a tag group and a tag may belong to multiple groups. The purpose of a tag group is to organize tags and a tag group may be associated to a category so that it can be automatically displayed in a context sensitive fashion when a customer is viewing products belonging to a specific category.

For example, the image below shows that the *MPAA Movie rating* and the *DVD Format* tag groups are displayed in the *DVD Movie>>Action* category . When a customer clicks on a tag , he refines the search to only show products that include that tag information. Multiple tags may be selected . When within the same tag group, they are OR'ed together. e.g. I want to see all movies with rating PG or PG-13. When within different groups they are AND'ed together. e.g. Show me all movies that have a PG or PG-13 rating and are available in Blu-ray format.

Tag functionality is available through the application API. The ProductSearch object has been enhanced to include an array of Tag Group objects, each of which may contain a number of tags. There is also a new API call `getTagGroupsPerCategory()` to enable you to determine which tag groups have been associated to a category. The Product object now has a tags attribute which is populated in the `getProduct()` method with an array of tags, so that you may see which tags have been associated to a product.

Managing Product Reviews

Product reviews may be managed through the Admin App where they can be edited, deleted and assigned different states. A review may be in one of 3 different states:

- *Visible state* - The review is visible in the storefront. i.e. The KonaCart engine API calls will return the review.
- *Invisible state* - The review is not visible in the storefront. i.e. The KonaCart engine API calls will not return the review.
- *Invisible Rejected state* - The review is not visible in the storefront. i.e. The KonaCart engine API calls will not return the review.

When a review is created by a customer, it may be assigned a default state which is defined by the configuration variable found in the Admin App under *Configuration >> Security and Auditing*. The default state should be either Visible or Invisible, depending on whether you intend to immediately display reviews or approve them before display. The procedure for approving them is to use the Reviews Panel in the Admin App and apply a search filter to display only invisible reviews. One by one, the review state may then be set to visible or rejected or the review may be deleted if you don't intend keeping inappropriate reviews. The reviews remain invisible when in the rejected state, but having them in this state informs you that they have been reviewed so that you can filter them out when reviewing a new batch of reviews.

Using CLOBs for Product Descriptions

To enable the use of CLOBs for product descriptions you need to modify the type of the "products_description" column in the "products_descriptions" table and set the "FETCH_PRODUCT_DESCRIPTIONS_SEPARATELY" configuration variable to "true". For convenience a little SQL script called database/Oracle/UseCLOBForProductDescriptions.sql is provided for this purpose.

The "Fetch Descriptions Separately" configuration variable can also be set in the Admin App.

In most situations you should leave this configuration variable set to "false" for better performance. The purpose is to modify the way product descriptions are retrieved from the database. If this is set to "true" the product descriptions are retrieved separately after the main product queries have been completed. This is required when using databases where the type of the products_description column has been changed from the KonaCart default value (for example to CLOB in Oracle) which cannot be used as part of the standard KonaCart product queries.

Credit Card Refunds

Real time credit card refunds through the payment gateway can be managed from the Admin App. You can also insert, edit and delete refund information in the database.

The code that performs the payment gateway credit transaction is within an AdminPayment class. An example of the AdminPayment class is provided for AuthorizeNet. It

must implement the com.konakartadmin.modules.AdminPaymentIf interface and by extending com.konakartadmin.modules.AdminBasePayment it inherits useful methods for sending the data to the payment gateway. The source code of all of these classes is supplied. Note that it is only possible to carry out a real time refund transaction if the payment gateway (during the original payment transaction) saves the transaction id and populates the Admin Payment Class attribute with the class name of the class that the Admin App can use to perform the refund. Again, a full example has been implemented for AuthorizeNet, and can be seen in the class com.konakart.actions.gateway.AuthorizenetAction.java

After concluding a payment gateway credit transaction, the order is placed either in a "Refund Approved" or "Refund Declined" state. From the Admin App you can decide whether a template based email should be sent directly to the customer with details of the transaction.

The above refund functionality is only available in the Enterprise version of KonaKart. For Community users a custom button has been introduced in the panel that processes returns. This feature allows you to extend the Admin App by adding your own administration functionality outside the standard Admin App. The following parameters are added to your defined URL to allow you to check security and/or operate on selected objects:

- *order_return_id* - id of the OrderReturn object
- *order_id* - id of the Order object
- *total_inc_tax* - total amount of the Order
- *sess* - session id
- *tx_id* - gateway transaction id

Note that this button will not be visible unless you define the label to be non-empty in the Admin App Configuration Panel.

Saving and Editing of Credit Card details

Configuration of Admin Application

The admin app now allows you to enter and/or edit credit card details for any order using the "edit order" panel, which is opened by selecting an order and clicking the edit button. This functionality is only available if the role allows it. In order to set privileges you must navigate to *Customers>>Maintain Roles* and select a role. Once the role has been selected you must click the Privileges button and set the check box on the Edit Order panel shown below.



Edit Customer	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Edit Order	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Edit Product	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

In order for the new role information to be picked up, you must log out and log in again, at which point in the edit order screen you should see the following fields (see below) which can be modified and saved. The information is saved in the database in fields e1 to e6 of the orders table in an encrypted format.

Order Status:	Pending	▼
Comments:	<div style="border: 1px solid #ccc; height: 60px; width: 100%;"></div>	
Notify Customer:	<input checked="" type="checkbox"/>	
Card Owner:		
Card Type:		Expires (MMYY):
Number:		CVV:
<input type="button" value="Save"/> <input type="button" value="Back"/>		

Configuration of Store Front Application

A new API call has been introduced in order to set the credit card details on an order.

```
/*
 * The credit card details in the CreditCard object passed in as a parameter, are saved in the
 * database for an existing order. Before being saved, this sensitive information is encrypted.
 * No update or insert is done for attributes of the CreditCard object that are set to null. The
 * credit card details are mapped as follows to attributes in the order object:
 *
 * e1 - Credit Card owner
 * e2 - Credit Card number
 * e3 - Credit Card expiration
 * e4 - Credit Card CVV
 * e5 - Credit Card Type
 *
 *
 * @param sessionId
 *          The session id of the logged in user
 * @param orderId
 *          The numeric id of the order
 * @param card
 *          CreditCard object containing the credit card details
 * @throws KKEception
 */
public void setCreditCardDetailsOnOrder(String sessionId, int orderId, CreditCardIf card)
    throws KKEception;
```

This API call can be integrated into the application where you see fit. i.e. You could write a simplified payment gateway that just collects the credit card information and saves it rather than sending it to a payment gateway. A configuration variable has been added which can be set in the admin app under *configuration>>Stock and Orders* .

Show Invisible Products	<input checked="" type="radio"/> true	<input type="radio"/> false
Store Credit Card Details	<input type="radio"/> true	<input checked="" type="radio"/> false
Stock Re-order level	5	

This config variable could be used in your code to determine whether or not to save the credit card details. Using AuthorizeNetAction as an example :

```
// Create a parameter list for the credit card details.
PaymentDetailsIf pd = order.getPaymentDetails();

String saveCCDetails = kkAppEng.getConfig(ConfigConstants.STORE_CC_DETAILS);
if (saveCCDetails != null && saveCCDetails.equalsIgnoreCase("true"))
{
    CreditCardIf cc = new CreditCard();
    cc.setCcOwner(pd.getCcOwner());
    cc.setCcExpires(pd.getCcExpiryMonth() + pd.getCcExpiryYear());
    cc.setCcNumber(pd.getCcNumber());
    cc.setCcCVV(pd.getCcCVV());
    cc.setCcType(pd.getCcType());
}

kkAppEng.getEng().setCreditCardDetailsOnOrder(kkAppEng.getSessionId(),
                                             order.getId(), cc);
}
```

Edit Order Number and Custom Fields

The admin app now allows you to enter and/or edit the order number and custom fields for any order using the "edit order" panel, which is opened by selecting an order and clicking the edit button. This functionality is only available if the role allows it. In order to set privileges you must navigate to *Customers>>Maintain Roles* and select a role. Once the role has been selected you must click the Privileges button and set the check box on the Edit Order panel shown below.

Edit Customer	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Edit Order	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Edit Product	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

In order for the new role information to be picked up, you must log out and log in again, at which point in the edit order screen you should see the following fields (see below) which can be modified and saved.

The screenshot shows a form for managing an order. At the top, there's a dropdown menu labeled "Order Status" with "Pending" selected. Below it is a large text area for "Comments". Underneath that is a section labeled "Notify Customer:" with a checked checkbox. The main body of the form contains several input fields: "Order #: 1232388300697", "Custom1:", "Custom2:", "Custom3:", "Custom4:", and "Custom5:". At the bottom right are two buttons: "Save" and "Back".

Shippers and Shipments

The Enterprise version of KonaCart allows you to define multiple shipments for an order so that this information is stored in the database and returned as an array of OrderShipment objects whenever a detailed order is retrieved through the APIs. The shipment information is displayed by default in the order details view of the storefront application and has been included in the velocity template for the eMail sent on an order state change (OrderStatusChange_3_en.vm) when the state changes to shipped. A similar template may be created for the partially shipped state.

Shipments

Date: 25/11/2013 11:15	Shipper: FedEx	Track
Item	Qty. Shipped	
Matrox G200 MMS - Model: Value - Memory: 4 mb	4	
SWAT 3: Close Quarters Battle	3	
Kindle Fire HD	2	

In order to configure the shipment functionality, the first step is to define one or more shippers under the Localizations >> Shippers section of the Admin App. Only the name (e.g. FedEx, UPS) is required although it may be useful to also enter the tracking URL template including the placeholder string {TRACK_NUM} which will be automatically substituted with the tracking number when a shipment is inserted. E.g.

https://tools.usps.com/go/TrackConfirmAction!input.action?iLabels={TRACK_NUM}

In this way the tracking URL can be made available to customers without having to enter it every time a new shipment is inserted.

Once the shippers have been defined, one or more shipments may be created for an order by clicking the Shipments button on the Orders panel after having selected an order. The online help of the Shipments panel provides more details regarding the process.

When in multi-vendor mode, whenever a shipment is inserted by a vendor, the information is automatically propagated to the parent order which is visible to the customer.

Wish Lists

Wish List functionality can be enabled in the *Configuration>>Store Configuration* section of the Admin App. Once enabled, the store front application will display an "Add to Wish List" button when viewing product details, and a Wish List tile containing products that have been added to the wish list. By default, wish lists are only available to logged in customers. However, a configuration variable, again in the *Configuration>>Store Configuration* section of the Admin App, may be set to enable wish list functionality for non logged in customers. In this case, when the customer logs in or registers, his temporary wish list is transferred to a permanent wish list belonging to the registered customer.

There is a Wish List screen that allows a customer to remove products from the list and also to transfer them directly to the shopping cart.

Gift Registries

Gift Registry functionality can be enabled in the *Configuration>>Store Configuration* section of the Admin App. Once enabled, the store front application will display Wedding List functionality. Note that it is relatively straightforward to customize the store front application to handle other types of gift registries such as birthday lists.

Only a registered customer can create a wedding list after logging into the application by clicking a link in the Wedding List section of the My Account page. The wedding list can be made public or private and once created, the shipping address and other details may be modified by clicking the edit link that will appear in the Wedding List section of the My Account page.

Products can be added to the wedding list by navigating to the product details page and clicking on the add to wedding list link. Note that you will see a link for each wedding list that you have created and the link will contain the name of the list. Once a product has been added to the list, you may modify the priority and the quantity desired attribute.

Any shopper can search for a public wedding list by clicking the Wedding Lists link at the top of the screen next to the Cart Contents and Checkout links. If no constraints are entered, all wedding lists are found and can be paged through. Otherwise constraints such as the wedding date or name of bride etc. can be used to narrow down the search. Once a wedding list has been found, you may click on its name in order to see the list of gifts sorted by priority. You may select one or more gifts and add them to the cart, ready for checkout. When gifts are added to the cart from this screen, the system will ensure that the wedding list is updated with quantity received (once the order has been paid for) and the default shipping address will be the address of the wedding list. During the checkout process, you can change the shipping address if you desire.

Gift Certificates

In KonaKart, a Gift Certificate is a product of type gift certificate, that can be bought by a customer and delivered as a digital download, through eMail or even regular mail. The gift certificate contains a promotion code which is usable only once to obtain a discount on an order.

Behind the scenes, a gift certificate object needs to be related to a promotion which can only be activated through the use of a coupon code. When an order containing one or more gift certificates is paid for, a coupon code and document must be created for each gift certificate. For example, the document could be a stylish pdf file containing the coupon code which the customer can download. The generation of the actual document is a customization that needs to be carried out, since the example source code just generates a simple txt file containing only the coupon code.

com.konakart.bl.OrderIntegrationMgr and *com.konakartadmin.bl.AdminOrderIntegrationMgr* are the two files that contain the code that is run when an order changes state after payment is received. The application code is run when the state is changed through the application (i.e. when a customer pays using a credit card) and the admin app code is run when the state is changed through the admin app. The actual method to look at is called *manageGiftCertificates()*. As you can see from the source code, this method implements the following tasks:

- Find the promotion attached to the gift certificate.
- Create a coupon code.
- Create a coupon with the new code.
- Attach the coupon to the promotion.
- Insert the coupon code into a downloadable product that can be downloaded by the customer. This becomes the gift certificate that the customer can forward on to the receiver of the gift..

Depending on your requirements, the method can be customized. For example, you may not wish the file to be made available as a digital download or you may wish to generate a more simple coupon code. An almost obligatory customization is the *getGiftCertificateFilePath()* method which at the moment produces a simple txt file just containing the coupon code. For production usage it should create maybe a pdf or html document that resembles a real gift certificate.

Enable Gift Certificates

In order to enable the gift certificate entry field in the storefront application, the configuration variable must be set in the Configuration>>Store Configuration section of the admin app.



Creating a Gift Certificate

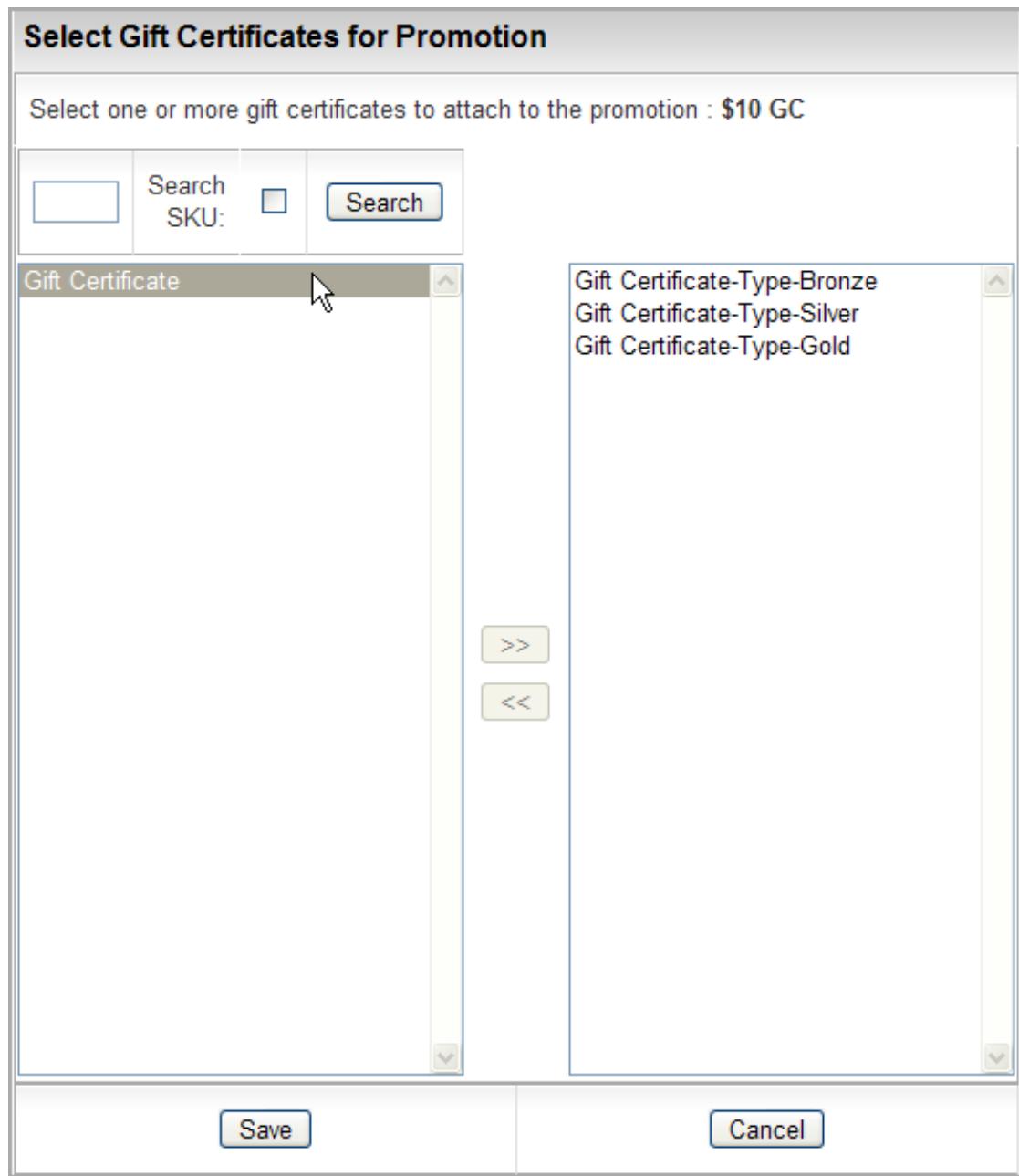
The following steps must be taken to create a gift certificate:

Create a product of type gift certificate using the admin app. You may choose different products for different amounts or choose a single product and configure the amounts as product attributes which can be selected by the customer before adding the certificate to the shopping cart.

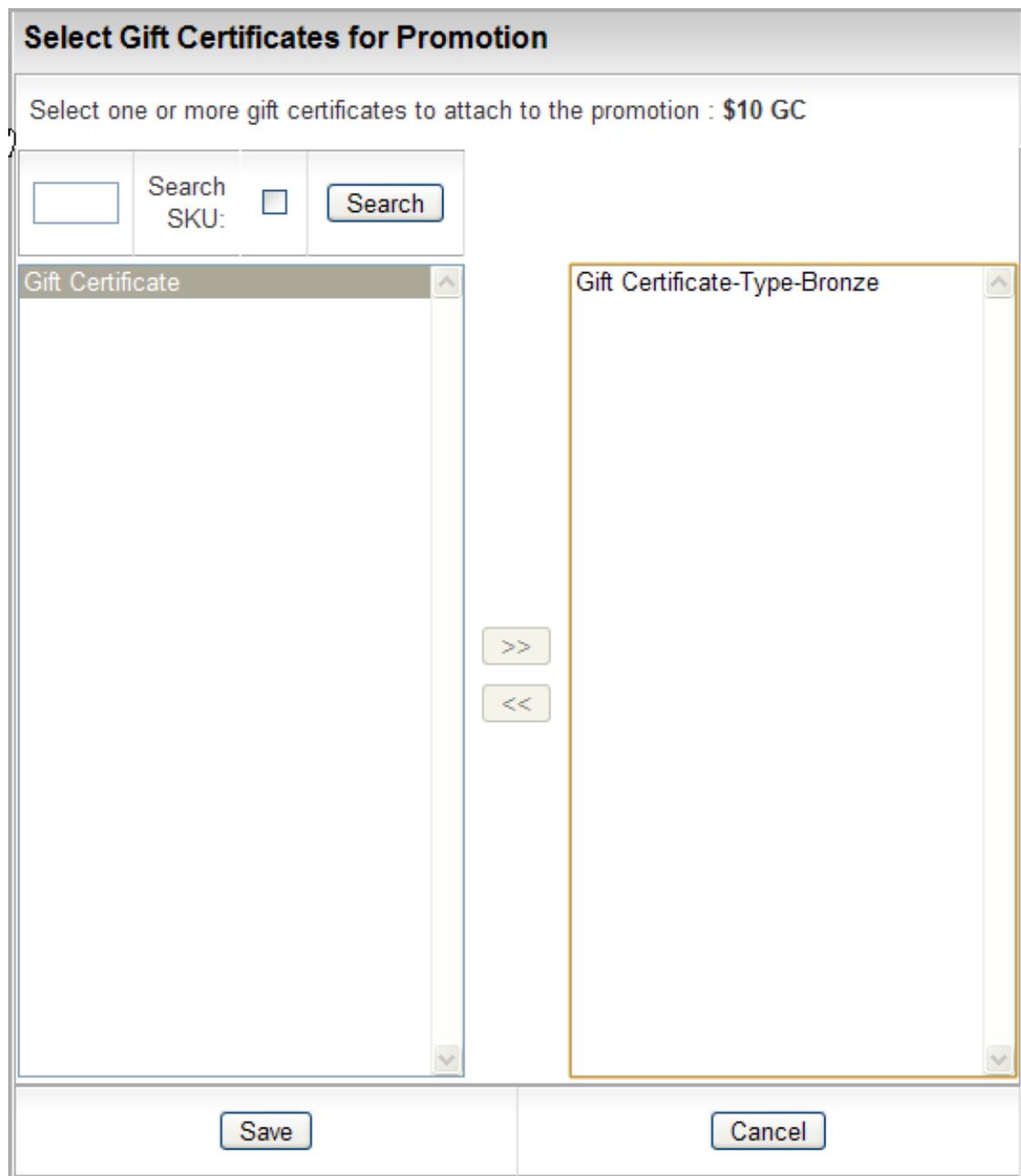
For each gift certificate, create a promotion of type Gift Certificate, where the value is equivalent to the value of the gift certificate. The promotion should require a coupon.

Name:	\$10 GC	Promotion Type:	Gift Certificate
Start Date:	Select date	End Date:	Select date
Active:	<input checked="" type="checkbox"/>	Requires Coupon:	<input checked="" type="checkbox"/>
Description:			
Minimum Order Value:	10	Min total quantity:	1
Min quantity for a product:	1	Value of Certificate:	10
Min order value before tax:	<input checked="" type="radio"/> true <input type="radio"/> false		
New Delete Save Cancel Coupons Rules Gift Certificates			

Once the promotion has been created you need to click the *Gift Certificates* button on the promotions panel in order to connect the promotion to a gift certificate.



The above example shows a Gift Certificate product that has three options (Gold, Silver, Bronze) which are expanded out when the product is selected. It is important the promotion is only associated with one of these options so the other two must be removed as shown below.



Note that a gift certificate product may be connected to any type of promotion, which means that you could, for example create a gift certificate that gives 10% discount for an order. Also the standard rules can be added to the promotion in order to filter it for certain products, categories, manufacturers, customers and expressions.

Creating a new Admin App User

New Admin App users can be created and configured using the Admin App, if you are logged in as a user with the required privileges.

Each KonaKart user can be assigned one or more roles in order to define the functionality available to that user. The first step is to create a new user in the Customers>>Customers section of the Admin App. The user type must be set to "Admin User".

Once the user has been created, roles may be assigned in the *Customers>>Assign Roles* section of the Admin App. The role assignment becomes active the next time the user logs in.

Creating New Roles

The default KonaKart database already contains a number of roles. New roles may be created (and existing roles may be edited) in the *Customers>>Maintain Roles* section of the Admin App.

Each role is mapped to a number of panels with a set of privileges. Each role to panel association may have a combination of read / insert / edit / delete privileges.

API call security may also be enabled in the *Configuration>>Security and Auditing* section of the Admin App. When enabled, this allows you to map roles to API calls and is useful for enforcing security through the SOAP Web Service interface of the Admin App.

Precise instructions on how to create new roles, can be found in the on-line help.

Default Customer Configuration

A Default Customer can be defined using the Admin Application. Only one default customer should be defined. It is a fictitious customer used to create a temporary order for displaying order totals before the checkout process. This is useful for example to create estimated shipping costs and promotional discounts (the order totals of the order), which can be displayed to a customer in the edit cart screen without him having to log in or register. The address of the default customer should be an address that will generate average shipping costs. i.e. If the majority of your business is national then it shouldn't be an international address.

The default customer is used by the application API method `createOrderWithOptions(String sessionId, BasketIf[] basketItemArray, CreateOrderOptionsIf options, int languageId)` if the options object is set to use the default customer. In this case the sessionId may be set to null.

Making Customers Invisible

By default when a customer is created, it is created as a "visible" customer. An attribute on the customer record called invisible is set to 0 to indicate that the customer is visible.

Also by default, an Admin App user with access to the customers panel will only be able to view and edit "visible" customers.

In some situations it might be useful to set a customer to be "invisible". This can be performed through the API when the customer is created using registerCustomer or for an existing customer using an edit-Customer API call.

When customers are made "invisible" they are not accessible from the Admin App unless the Admin User is assigned a special privilege that allows him to access both visible and invisible customers.

To allow an Admin User to access both visible and invisible customers you need to set the custom1 flag on the "Customers 2" panel for the required role on the Privileges panel as illustrated below:

Panel Name	Insert	Edit	Delete	Cust1	Cust2	Cust3
Select All	<input type="checkbox"/>					
Change Password	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Customer Addresses	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Customer Communications	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
Customer Groups	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Customer Orders	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Customers	<input checked="" type="checkbox"/>					
Customers 2	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Edit Customer	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>
Insert Customer	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Manufacturer Addresses	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Product Addresses	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Reward Points Configuration	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Save Cancel

Unlike the "Customers" panel, the "Customers 2" panel shown above is not a real panel but a "virtual" panel which only exists for the purpose of setting this privilege to access invisible customers. Additional privileges may be added to this virtual panel in the future.

Customer Groups

Customer Group functionality has been available in KonaKart since Version 2.2.3.0.

A customer group is a way of aggregating customers that are similar in some way. For example, you may use them to distinguish between retail and wholesale customers or between company employees and external customers etc.

Customer groups may be created and maintained using the KonaKart Admin application. Once a group has been created, you can associate a customer to that group through a drop list in the Edit Customer panel. When editing a customer, if the Customer group is changed to a valid new group, you will be prompted to send a template based email to the customer. This is a useful feature for when the customer is going through an approval process. For example, a customer may have registered through the application as a wholesale customer. During registration he was placed in a "Waiting for Approval" customer group and now the administrator may approve or decline the request. As a result of the approval, the customer may receive an email informing him of the decision. The template used is in the form *CustGroupChange_groupId_lang.vm* e.g. *CustGroupChange_2_en.vm*, if the customer has been moved to the group with id equals 2 in a system where the language code is "en". This means that different templates can be used depending on which group the customer has been moved to. i.e. You could have different templates for approved and denied requests.

Groups may be used to:

- Control what prices are displayed to customers.
- Each group has a PriceId attribute which may have a value of 0 to 3. When set to 0, a customer belonging to that group will see the normal price of the product (i.e. the price attribute). When set to 1, the customer will see the price defined in the Price 1 attribute of the product and so on for 2 and

3. This functionality for example, allows you to display wholesale prices to wholesale customers and retail prices to retail customers. If a customer belongs to no groups, then the price from the normal price attribute is displayed. Note that in the Admin App you may change the price labels from Price1, Price2 etc. to a more meaningful description such as Wholesale price, MRP, Employee price etc.

- Enable promotions.
 - Promotions may be enabled for customers belonging to a particular group. i.e. You may want to enable certain promotions just for your retail customers.
- Send communications.
 - Bulk emails may be sent to only customers belonging to a particular customer group.

Auditing

Auditing can be enabled on the Admin App API in order to keep track of when API calls are made and by whom. All audit data is written to the KonaKart database and can be viewed in the *Audit Data>>Audit Data* section of the Admin App.

Auditing can be configured in the *Configuration>>Configure Auditing* section of the Admin App. It can be configured independently for Reads / Edits / Inserts and Deletes. The level may be set to "summary" or "detailed" :

- Summary : The name of the API call, the type of Action (read , write etc.) , the id of the object being edited, inserted etc. (where applicable) and the time stamp are saved.
- Detailed : Where possible, the objects being deleted, edited etc. are also saved in a serialized form . Note that enabling detailed auditing can have a detrimental affect on performance and may save a large amount of data.

Custom Credential Checking

When the `login()` method is called, KonaKart instantiates a class defined by the property `LOGIN_INTEGRATION_CLASS` . If this property isn't set, the class that is instantiated is `com.konakart.bl.LoginIntegrationMgr` . If you write a custom class it must implement the interface `com.konakart.bl.LoginIntegrationMgrInterface` which contains the method:

```
public int checkCredentials(String emailAddr, String password)
    throws KKException;
```

The `checkCredentials()` method can return the following values:

- A negative number in order for the login attempt to fail. The KonaKart `login()` method will return a null sessionId.
- Zero, to signal that this method is not implemented. The KonaKart `login()` method will perform the credential check.
- A positive number for the login attempt to pass. The KonaKart `login()` will not check credentials, and will log in the customer, returning a valid session id.

A similar mechanism exists for the Admin App. The property is called `ADMIN_LOGIN_INTEGRATION_CLASS` and if it isn't set then the class that is instantiated is

`com.konakartadmin.bl.AdminLoginIntegrationMgr`. If you write a custom class it must implement the interface `com.konakartadmin.bl.AdminLoginIntegrationManagerInterface` which contains the method as described above.

```
public int checkCredentials(String emailAddr, String password)
throws KKAdminException;
```

This mechanism is a useful generic way to implement a Single Sign On system or just to implement your own custom credentials checking..

The `LOGIN_INTEGRATION_CLASS` and `ADMIN_LOGIN_INTEGRATION_CLASS` properties can be edited in the *Configuration>>Security and Auditing* section of the Admin App.

Custom Credential Checking - LDAP

The Enterprise version of KonaKart contains an LDAP module which can be configured through the Admin App. As can be seen from the image below, once installed, the module may be enabled / disabled by setting the status to true or false.

Other Modules	
Module Name	Sort Order
LDAP	0
<input type="button" value="Install"/> <input type="button" value="Remove"/>	
LDAP Status	<input checked="" type="radio"/> true <input type="radio"/> false
Sort order of display	0
LDAP App Mgr Class Name	com.konakart.bl.LDAPMgr
LDAP Admin Mgr Class Name	com.konakartadmin.bl.AdminLDAPMgr
LDAP User Name	uid=admin,ou=system
LDAP Password	*****
LDAP URL	ldap://localhost:10389
Person Object DN	ou=people,dc=example,dc=com
<input type="button" value="Save"/> <input type="button" value="Cancel"/>	

The LDAP user name, password and URL are used to connect to the LDAP directory. The defaults shown above are those for Apache Directory. The Person Object DN is the distinguished name for accessing a person object within the LDAP directory. The default value of "ou=people,dc=example,dc=com" is valid for the example LDIF file which can be found in the KonaKart/custom/modules/src/com/konakartadmin/modules/others/ldap file system directory after installing KonaKart. This file contains a couple of entries (people) which match the default Person Object DN and can be used to test the LDAP module. For production use, the Person Object DN will need to be changed to match the tree structure of your LDAP directory.

The source of the code that connects to LDAP and verifies the customer's credentials can be found in the file KonaKart/custom/appnEE/src/com/konakart/bl/LDAPMgrCore.java in the method:

```
public int checkCredentials(String emailAddr, String password) throws Exception
```

This code may be modified and compiled in order to apply specific logic for the structure of your LDAP directory. The code has many comments and debug statements in order to be self-explanatory. As in the case of the LoginIntegrationMgr the checkCredentials() method can return the following values:

- A negative number in order for the login attempt to fail. The KonaKart login() method will return a null sessionId.
- Zero, to signal that this method is not implemented. The KonaKart login() method will perform the credential check.
- A positive number for the login attempt to pass. The KonaKart login() will not check credentials, and will log in the customer, returning a valid session id.

The Admin App allows you to define a class name for the LDAP object that gets instantiated for both the application engine and the admin engine whenever a person attempts to log in. The default class for the application engine is *com.konakart.bl.LDAPMgr* and can be found in the KonaKart/custom/appnEE/src/com/konakart/bl directory. The class for the admin engine is *com.konakartadmin.bl.AdminLDAPMgr* and can be found in the KonaKart/custom/adminappnEE/src/com/konakartadmin/bl directory . They both implement interfaces and so can be substituted with other classes that implement the same interfaces. These classes only contain a small amount of code that reads the configuration variables to set up the LDAP module, The code that actually makes the connection (and validates the credentials) can be found in KonaKart/custom/appnEE/src/com/konakart/bl/LDAPMgrCore.java and is common to both the application and admin engines.

Multi-Store Configuration and Administration

Multi-Store functionality is provided as part of the KonaKart Enterprise Extensions.

Before Multi-Store can be configured it must be installed by following the instructions for installing the KonaKart Enterprise Extensions .

Introduction

Since version 3.0.1.0, the enterprise version of KonaKart can run in multi-store mode. When in this mode, one instance of the KonaKart engine can manage multiple stores. Previous to this version, each store would have required its own instance of the KonaKart engine. Each store still requires its own database schema (i.e. its own instance of database tables).

Configuring KonaKart to function in Multi-Store Mode

The instructions that follow, are to set up two stores on localhost using the Tomcat servlet engine in the download kit. Obviously this isn't a production scenario, but will quickly allow you to use two KonaKart stores picking up separate data but running on a single KonaKart deployment. The URLs to access the stores will be :

- <http://store1.localhost:8780/konakart/Welcome.action>
- <http://store2.localhost:8780/konakart/Welcome.action>

The first step is to install the KonaKart Enterprise Extensions. It is recommended that you use the automated GUI installation but it is also possible to install manually. During the installation process you are prompted to choose one of the multi-store modes that KonaKart supports. All of the modes are compatible with the following steps that need to be taken.

Configuration Steps

Mapping

DNS has to be set up to map store1.localhost and store2.localhost to 127.0.0.1. This can be done by editing your hosts file as follows:

```
127.0.0.1      store1.localhost
127.0.0.1      store2.localhost
```

Note that this is a manual configuration step that is not carried out by the Enterprise Extensions installation.

In the Tomcat server.xml file you must set up aliases in the Host section:

```
<Host name="localhost" appBase="webapps"
      unpackWARs="true" autoDeploy="true"
      xmlValidation="false" xmlNamespaceAware="false">
    <Alias>store1.localhost</Alias>
    <Alias>store2.localhost</Alias>
</Host>
```

BaseAction.java

This step that is not carried out automatically by the Enterprise Extensions installation and so needs to be performed manually.

com.konakart.actions.BaseAction.java contains a method called `getstoreIdFromRequest()`. The purpose of this method is to derive the storeId from the `HttpServletRequest` so that the correct storeId can be passed to the KonaKart engine when it is instantiated. Below you can see an example where the store id is derived from the server name. It returns store1 when the server name is store1.localhost and store2 when the server name is store2.localhost. You can find the source for this class in the `KonaKart\custom\appn\src\com\konakart\actions` directory. All you need to do is to un-comment the code and compile it. In other parts of this document there are instructions for compiling and deploying the modified code. There is an ant build file in the `KonaKart\custom` directory with suitable targets for building and deploying the modified jars.

```

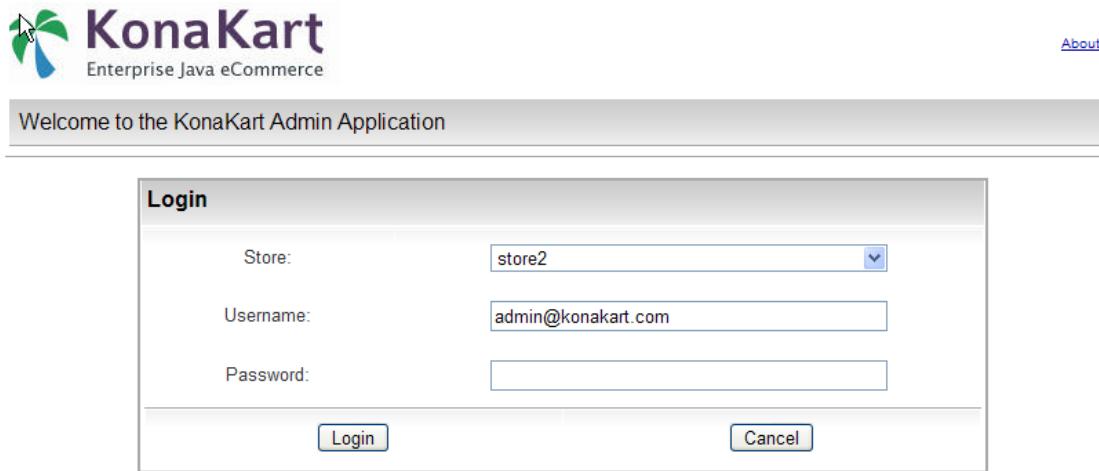
/**
 * In a multi-store scenario, you should implement your own logic here to extract the storeId
 * from the request and return it.
 *
 * @param request
 * @return Returns the storeId
 */
private String getStoreIdFromRequest(HttpServletRequest request)
{
    /*
     * If the server name could contain store1.localhost or store2.localhost which both point to
     * localhost, we could get the store id from the server name.
     */
    if (request != null && request.getServerName() != null)
    {
        String[] nameArray = request.getServerName().split("\\\\.");
        if (nameArray.length > 1)
        {
            String storeId = nameArray[0];
            return storeId;
        }
    }
    return null;
}

```

Final Steps

Once the modifications have been completed, you must re-start Tomcat. Assuming the database tables for store1 and store2 both contain valid data, you should be able to navigate directly to <http://store1.localhost:8780/konakart/Welcome.action> [<http://store1.localhost:8780/konakart/Welcome.action>] and to <http://store2.localhost:8780/konakart/Welcome.action> [<http://store2.localhost:8780/konakart/Welcome.action>].

In order to configure KonaCart separately for each store, the Admin App now allows you to choose the store before logging in.



Choose store during login

Once you've logged in you will be able to configure the store that was chosen from the drop down list.

Multi-Store Configuration

This section describes the functions and configuration options available to a KonaKart administrator once Multi-Store has been successfully installed.

All functionality related to the creation of new stores is only relevant if the Multi-Store Single Database Mode is selected.

The administration panels are made accessible to an Admin App user by virtue of role settings that have to be applied by a Super User of the KonaKart system. By default, following a successful installation of the Enterprise Extensions, the "Super User" user(s) created will have access to the panels described below.

By contrast, the panels discussed below will not be visible in the Admin App (by default) in Single-Store Mode, or Multi-Store Multiple DB modes where the functionality is not available.

Multi-Store Management Panel

Assuming you have sufficient privileges, you can maintain multiple stores on the Multi-Store Management Panel:

Store Id	Store Name	Enabled	Maintenance	Deleted
store1	store1	✓	✗	✗
store2	store2	✓	✗	✗

KonaKart Admin Application - Store Maintenance

If you have been granted sufficient privileges, you may search for a store in the mall by entering any combination of the search criteria at the top of the panel then clicking the Search button.

The search is not case sensitive unless your database is configured to be non-case sensitive. Wildcard configuration parameters control the precise searches of substrings (See *Configuration>>Admin App Configuration*) but with the default settings a wildcard is added before and after all search text which means that, for example, storeId *StoreX* will be returned if you enter *tor* as the StoreId.

In a similar fashion you can search for a store in the mall on whether or not it is enabled or disabled, under maintenance or not under maintenance and whether or not it is marked as deleted.

Only the stores you are authorized to maintain will be returned.

Any displayed stores may be edited or deleted by selecting them in the table then clicking the respective buttons at the bottom of the list assuming the relevant privileges have been granted to the current user.

The user must have edit store privileges for the Edit Store panel in order for the Edit button to be visible.

Creating a New Store - Without Cloning

Note that storeIds must contain no embedded spaces and no special characters that cannot be used when creating directory names.

To create a new store without cloning click the *New* button whereupon a new panel will be displayed on which you define the attributes for the new store.

Stores can have a number of different statuses. These are defined as follows:

Enabled = active stores that are available for use. If disabled a store is not accessible to application or admin app users.

Under Maintenance = stores that are currently being maintained so are inaccessible to application users, but can be accessed by admin app users.

Deleted = stores that are deleted are no longer available for use by application or admin app users. These stores are not physically deleted from the database but only marked as "deleted". Therefore it is possible to "Un-delete" a store if required simply by changing the deleted status back to false.

When a new store is created, the following new users are created:

- Multi-Store Single Database - Shared Customers Mode:

One user is created with the Store Administrator's role.

Username	Password	Roles
{new store Id}-admin@konakart.com	princess	KonaKart Store Administrator

- Multi-Store Single Database - NON-Shared Customers Mode:

One user is created with the Store Administrator's role, and one user with the Super User role.

Username	Password	Roles
{new store Id}-admin@konakart.com	princess	KonaKart Store Administrator
{new store Id}-super@konakart.com	princess	KonaKart Store Super User

The users created above will have the {new store Id} replaced with the storeId of the new store. Thus, if the new store was called "store3" the new users would be named *store3-admin@konakart.com* and *store3-super@konakart.com* as applicable.

The particular role that is assigned to the each user can be defined in the Admin App. You can configure the roles themselves as you wish so that when new stores are created, the users that are created will have only the permissions that you have defined.

In the case of the Non-Shared Customers Mode, where a new "Super User" is created, you can decide to provide these credentials to a "Super User" of the new store or keep them secret.

Creating a New Store - With Cloning

To create a new store by cloning another store, select the row containing the store you wish to clone and click the *Clone* button whereupon a new dialogue will be displayed on which you define the attributes for the new store.

For cloning you are prompted for the credentials of a user authorized to export data from the store to be cloned.

Differences between "New" and "Clone"

Both the *New* and *Clone* options can be used to create a new store. The major difference between the two options is that in the case of the clone, a lot more data is cloned for the new store.

When you create a new store using the "New" option, the following objects are created in the new store:

- Standard users and their respective role assignments (see above).
- Address formats, Countries, Languages, Tax zones, Currencies.
- Order Statuses, Configuration Data.

When you clone a store, using the "Clone" option, in addition to the above, the following objects are exported from the store to be cloned and imported into the new store:

- Products, Categories, Manufacturers, Coupons
- Customer Groups
- Zones, Sub-Zones, Tax Classes, Tax Rates.
- Tags and Tag Groups.

Note that cloning does not include the copying of the following objects:

- Auditing Data, IPN (payment gateway responses).
- Customers, Reviews
- Configuration Groups (this table is no longer used by KonaKart)

You can edit stores using the Edit Store Panel:

KonaKart Admin Application - Edit or Insert Store

Multi-Store Table Sharing

These details will not be of interest to many KonaKart Administrators as they will regard this as an implementation detail but it could be useful when analyzing the database directly for various purposes.

When in MultiStore Single DB Mode (engine mode 2) database tables are shared between stores. In most cases the data is segmented for each store using a store_id column to identify which store the data belongs to. Some tables are shared across all stores so no store_id column is created, whereas some tables are shared only in customer shared or products shared mode.

When tables are "shared" there is either no store_id column on the table to identify which store the data belongs to, or if the store_id column is present, it isn't used.

<i>Tables that are Always Shared</i>	<ul style="list-style-type: none"> • configuration_group • kk_api_call • kk_cookie • kk_panel
--------------------------------------	---

	<ul style="list-style-type: none"> • kk_role • kk_store • utility
<i>Tables Shared only when Customers are Shared</i>	<ul style="list-style-type: none"> • address_book • address_format • countries • customers • customers_info • geo_zones • kk_customer_group • kk_customers_to_role • kk_role_to_api_call • kk_role_to_panel • sessions • tax_class • tax_rates • zones • zones_to_geo_zones
<i>Tables Shared only when Products are Shared</i>	<ul style="list-style-type: none"> • kk_payment_schedule • kk_tag • kk_tag_group • kk_tag_group_to_tag • kk_tag_to_product • languages • manufacturers • manufacturers_info • orders_status • products • products_attributes • products_description

- products_options
- products_options_values
- products_options_values_to_products_options
- products_attributes_download
- reviews
- reviews_description
- specials

Multi-Store Configuration Panel

You can configure the following parameters on the Multi-Store Configuration Panel:

Welcome back, admin@konakart.com (store1)
[Set Language](#) [Change Password](#) [Sign Out](#) [About](#)

Multi-Store Configuration	
Multi-Store Template Store	store1
Multi-Store Admin Role	Store Administrator
Multi-Store Super User Role	Super User
Admin Store Integration Class	com.konakartadmin.bl.AdminStoreIntegrationMgr
KonaCart new store creation SQL	C:/Program Files/KonaKart/database/MySQL/konakart_new_store.sql
User new store creation SQL	C:/Program Files/KonaKart/database/MySQL/konakart_user_new_store.sql
<input type="button" value="Save"/> <input type="button" value="Cancel"/>	

KonaKart Admin Application - Multi-Store Configuration Parameters

These are the configuration variables that are used in a multi-store installation.

You can define the multi-store template storeId to the store whose reports you wish to copy for new stores (in the future, more configuration data may be copied from the template store).

You can define the roles that will be assigned to new users that are created when you create new stores. The number of new users that are created when creating new stores is dependent on whether you are using Shared or Non-Shared Customers. In Shared Customers mode, only a "Store Administrator" user is created, but in Non-Shared Customers mode a "Super User" user is also created. You can change the roles that are assigned for the users and change the roles themselves to configure the permissions granted to new users exactly as you please.

When new stores are created some SQL commands are run to set up the database in various ways. First the "KonaKart new store creation SQL" script is run, then KonaKart loads some set-up data that isn't in any SQL script, then finally this is followed by the execution of the "User new store creation SQL" script. If you need to change the data that is loaded, you should add your SQL commands to the "User new store creation SQL" script and not modify the "KonaKart" script.

Note the following default SQL file names: (If you set up your Multi-Store configuration manually rather than using the automatic installation, you will need to set these up appropriately for your chosen mode. These are set on the Multi-Store Configuration panel for the value labeled "KonaKart new store creation SQL").

- konakart_new_store.sql - for multi-store without sharing customers
- konakart_new_store_cs.sql - for multi-store with shared customers

For more specialized requirements there is a "hook" function that is called whenever a store is created or modified. If you need special behavior at these points, you have to implement this class in java (default name is `com.konakartadmin.bl.AdminStoreIntegrationMgr`) and it will be executed by KonaKart at the appropriate times. Your implementation must implement the `com.konakartadmin.blif.AdminStoreIntegrationMgrInterface` interface which has the following methods:

```
package com.konakartadmin.blif;

import com.konakartadmin.app.AdminStore;

/**
 * Used to provide an integration point when a store is added or changed
 */
public interface AdminStoreIntegrationMgrInterface
{
    /**
     * Called whenever a new store is added
     *
     * @param store
     *          The new store
     */
    public void storeAdded(AdminStore store);

    /**
     * Called whenever a new store is added
     *
     * @param oldStore
     *          The store object before the change
     * @param newStore
     *          The new store object after the change
     */
    public void storeChanged(AdminStore oldStore, AdminStore newStore);
}
```

If required, you can change the name of the class that is instantiated so long as it implements `com.konakartadmin.blif.AdminStoreIntegrationMgrInterface` and you modify its name in the configuration panel in the *Admin Store Integration Class* field.

Product Synchronization

When running in multi-store shared product / shared category mode, the stores may be used to provide pre-production environments where products may be inserted and edited without being made live. This means that sometimes it is useful to run KonaKart in this mode even when you are managing a single store.

The Synchronize Products panel in the Admin App allows you to copy products from the current store to any other defined store. The copy process creates a new product by cloning the current one and inserts it in

the destination store. When using KonaCart to provide staging environments, all products must be present in only one store. i.e. You mustn't associate a product with more than one store.

<u>Id</u>	<u>SKU</u>	<u>Product Name</u>
1		Matrox G200 MMS
2		Matrox G400 32MB
3		Microsoft IntelliMouse Pro
4		The Replacement Killers
5		Blade Runner - Director's Cut
6		The Matrix
7		You've Got Mail
8		A Bug's Life
9		Under Siege
10		Under Siege 2 - Dark Territory
11		Fire Down Below

Displaying 1 to 11 (of 29 Products) Page of 3 [→](#)

[Copy](#) [Copy All](#) [Edit](#)

KonaCart Admin Application - Synchronize Products

When searching for products to synchronize, you select from the drop list the store that you want to synchronize with and then from the other drop list you may choose to search for new products or products which are out of sync. New products are products that exist in the current store but have never been copied to the destination store. Out of sync products are products that exist in both stores but have different last modified dates. If you are searching for a single product, you may enter the product id or sku as one of the search filters.

In the case of new products, when you select one and click the copy button, it creates a copy and inserts it into the destination store. When the copy button is clicked for an out of sync product, the existing product in the destination store is edited rather than a new one being inserted. In some cases the product may have associated products such as up-sell, cross sell or bundled products that don't exist in the destination store. The default functionality is to copy over these products as well. You may decide not to copy them by unchecking the check box in the confirmation panel.

When you click the Copy All button, all products (new and out of sync) are copied to the destination store using the batch program in the com.konakartadmin.bl.AdminProductBatchMgr called synchronizeStores-Batch. A log file is created in the log directory defined in the Admin App under Configuration >> Logging. The program runs as a background task and progress may be monitored by clicking on the search button to display the products that haven't been synchronized yet.

When a product is copied from one store to another store, the copied product has a different product id. However each KonaCart product has a UUID attribute whose value is identical for all copies of the

same product. There is also a last modified date attribute which is used to determine whether products with identical UUIDs in different stores are synchronized or not. These two attributes allow KonaKart to determine whether a product already exists in the destination store and if it does exist, whether it is in sync.

Scheduling in KonaKart

Job scheduling in KonaKart is achieved with an integration with the Open Source Quartz scheduler. (It's also easy to use any other scheduler if you prefer but the notes below refer to integration of the Quartz scheduler). Quartz integration is provided in the Enterprise Extensions of KonaKart.

Configuring Quartz to execute KonaKart jobs

This section describes how to configure Quartz to execute KonaKart batch jobs. All the files mentioned are provided as examples in the Enterprise Extensions of KonaKart.

The quartz jar (version 2.1.7 is provided in KonaKart v7.2.0.0) is provided in the kit and is added to the konakartadmin/WEB-INF/lib directory alongside all the other KonaKart Admin jars.

Configuring web.xml

To start-up Quartz you need to uncomment a section in the konakartadmin webapp's web.xml file (this is located in the konakartadmin/WEB-INF directory).

You need to uncomment this section so that the QuartzInitializerServlet starts up when KonaKart starts.

```
<!-- Quartz Scheduler
<servlet>
    <servlet-name>QuartzInitializer</servlet-name>
    <servlet-class>org.quartz.ee.servlet.QuartzInitializerServlet</servlet-class>
    <init-param>
        <param-name>shutdown-on-unload</param-name>
        <param-value>true</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
</servlet>
End of Quartz Scheduler -->
```

Configuring quartz.properties

The Quartz properties file should be placed on the konakartadmin classpath (the installer will place it in the konakartadmin/WEB-INF/classes directory).

The following is the default for KonaKart:

```

=====
# Configure Main Scheduler Properties
=====

org.quartz.scheduler.instanceName = KonaKartScheduler
org.quartz.scheduler.instanceId = AUTO

=====
# Configure ThreadPool
=====

org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool
org.quartz.threadPool.threadCount = 3
org.quartz.threadPool.threadPriority = 5

=====
# Configure JobStore
=====

org.quartz.jobStore.misfireThreshold = 60000
org.quartz.jobStore.class = org.quartz.simpl.RAMJobStore

=====
# Configure Plugins
=====

org.quartz.plugin.triggHistory.class = org.quartz.plugins.history.LoggingJobHistoryPlugin

org.quartz.plugin.jobInitializer.class = org.quartz.plugins.xml.XMLSchedulingDataProcessorPlugin
org.quartz.plugin.jobInitializer.fileNames = konakart_jobs.xml
org.quartz.plugin.jobInitializer.failOnFileNotFound = true
org.quartz.plugin.jobInitializer.scanInterval = 60
org.quartz.plugin.jobInitializer.wrapInUserTransaction = false

# No longer used
#org.quartz.plugin.jobInitializer.overWriteExistingJobs = true

```

A useful parameter that you may wish to modify is the fileNames parameter which, by default, is set to "konakart_jobs.xml". The "konakart_jobs.xml" file contains definitions of the example KonaKart jobs that can be run.

It is possible that you may wish to have additional job definition files like "konakart_jobs.xml" for defining batch schedules for other stores in a multi-store environment. Additional job definition files can be added to the filenames property.

Configuring konakart_jobs.xml

The default job definition file is called "konakart_jobs.xml" and is placed in the konakartadmin/WEB-INF/classes directory.

The following is an extract from "konakart_jobs.xml" that defines a single job:

```

<job>
  <name>RemoveExpiredCustomers</name>
  <group>KonaKart Batch Jobs</group>
  <description>Removes expired Customers for privacy and performance</description>
  <job-class>com.konakartadmin.bl.ExecuteBatchEE</job-class>
  <durability>false</durability>
  <recover>false</recover>
  <job-data-map>
    <entry>
      <key>credentialsFile</key>
      <value>konakart_jobs.properties</value>
    </entry>
    <entry>
      <key>executionClass</key>
      <value>com.konakartadmin.bl.AdminCustomerBatchMgr</value>
    </entry>
    <entry>
      <key>executionMethod</key>
      <value>removeExpiredCustomersBatch</value>
    </entry>
    <entry>
      <key>param0</key>
      <value>removeExpiredCustomers</value>
    </entry>
    <entry>
      <key>param1</key>
      <value>true</value>
    </entry>
    <entry>
      <key>param2</key>
      <value>25</value>
    </entry>
    <entry>
      <key>param3</key>
      <value>0-1-2-3</value>
    </entry>
  </job-data-map>
</job>

```

This particular batch job removes expired sessions from KonaKart. Note the following parameters:

job-class = com.konakartadmin.bl.ExecuteBatchEE

The job-class defines the class that Quartz will execute when the job is run. The ExecuteBatchEE class is the Quartz > KonaKart bridge class that can be used to call KonaKart batch jobs from jobs defined in Quartz.

Another job-class that can be used here is the com.konakartadmin.bl.ExecuteMultiStoreBatchEE job class. This Quartz > KonaKart bridge class executes the specified job in a loop, once for every enabled store. Note that this job will use the credentials specified in the konakart_jobs.properties file to access each store in turn so these credentials need to be valid for each store.

key: credentialsFile = konakart_jobs.properties

Inside the job-data-map tag an entry is defined that contains the name of a KonaKart "Credentials" file. The file can be named anything you like but it must be located on the class path so that ExecuteBatchEE can access it. In our example, the file is called "konakart_jobs.properties" (see below for more detail on the contents of this file). The credentials are required so that the ExecuteBatchEE class can create and log into a KonaKart Admin Engine, then finally execute the "execute()" API call on that engine to run the defined batch job.

key: executionClass = com.konakartadmin.bl.AdminCustomerBatchMgr

The execution class is the name of the class that will be instantiated by KonaKart to execute the defined batch job. It is possible to define your own class as the executionClass which is an excellent way to extend KonaKart's batch functionality.

key: executionMethod = removeExpiredCustomersBatch

The execution method is the name of the method on the executionClass that will be executed by KonaKart to run the defined batch job. It is possible to define your own class and methods as described above to extend KonaKart functionality as you require.

key: param0, param1, ..., paramN

The "param" keys are the parameters that are passed to the batch job. These can be whatever the batch job needs but must be represented as strings and the ordering is significant.

It is essential that you name the parameters "param0", "param1" and so on for each of the parameters.

In our example, param0, the first argument, is used in this job to define the name of the log file produced by the batch job.

If you wish to pass the session Id of the logged-on user who will execute the job you can define a parameter with the special value *SESSION_ID*. If KonaKart sees a parameter with this value it will replace it with the session Id used to run the execute() API call which is used by the KonaKart batch architecture. Other than this runtime replacement the parameter is identical to the other parameters so needs to be defined on your method signature in the correct position. If in doubt check the examples under the batch directory of your installation which can be found under the custom directory at the top level of your KonaKart home directory. Full source code is provided (Enterprise Only) for the example batch jobs such as in * \KonaKart\custom\batch\src\com\konakartadmin\bl\AdminCustomerBatchMgr.java.

Quartz executes the defined jobs when defined "Triggers" fire. These triggers are also defined in the konakart_jobs.xml file:

The following is an extract from "konakart_jobs.xml" that defines a single trigger:

```
<trigger>
  <cron>
    <name>RemoveExpiredCustomersTrigger</name>
    <group>KonaKart Triggers</group>
    <description>Trigger for RemoveExpiredCustomers</description>
    <job-name>RemoveExpiredCustomers</job-name>
    <job-group>KonaKart Batch Jobs</job-group>
    <!-- every hour          0 0 * ? * * -->
    <!-- every 30 seconds   0,30 * * ? * * -->
    <!-- every day @ 9:00pm  0 0 21 ? * * -->
    <cron-expression>0 5 21 ? * *</cron-expression>
  </cron>
</trigger>
```

This is where you define when you want a particular batch job to be executed.

In this case we have chosen to use the cron trigger definition but Quartz provides other trigger mechanisms that you can easily use instead if you prefer.

Some examples of cron expressions are provided (commented out).

Configuring konakart_jobs.properties

The default job definition credentials file is called "konakart_jobs.properties" and is placed in the konakartadmin/WEB-INF/classes directory.

The file name is actually defined in the "konakart_jobs.xml" file - see the credentialsFile tag above. Hence, you can easily change the name to suit your needs. You can also have more than one credentials file - which would be required if you needed to run batch jobs on different stores, in a multi-store configuration, where the access credentials were different for the different stores.

The following is an extract from "konakart_jobs.properties" that defines the parameters required to create and log in to an Admin Engine:

```
#=====
# konakart_jobs.properties
#
# Used to define the engine configuration for batch jobs.
# If multiple configurations are required, use multiple files and define
# these in multiple org.quartz.plugin.jobInitializer.fileNames files.
#=====

# -----
# Engine mode that the batch jobs will use
# 0 = Single Store (default)
# 1 = Multi-Store Multiple-Databases (add konakart.databases.used above as well)
# 2 = Multi-Store Single Database

konakart.mode = 2

# -----
# When in multi-store single database mode, the customers can be shared between stores
konakart.customersShared = true

# -----
# Credentials to use for the batch jobs

konakart.user      = admin@konakart.com
konakart.password = princess
```

It is recommended that you secure this file appropriately as it could contain the credentials of a privileged Admin user.

Other configuration options are possible in credentials files (eg. the engine class can be defined and the properties files it uses can be defined). If the other configuration parameters are left commented out default values are used.

Customizing the KonaKart jobs

This section describes how to customize the batch jobs provided or create new ones and rebuild them. Source and build files are provided in the Enterprise Extensions of KonaKart.

The java source files can be found under the KonaKart installation directory under *custom/batch* . The java files can be modified here and new ones added as required to implement different or new batch functionality.

To rebuild the *konakartadmin_batch.jar* (the jar containing all of the batch class files) execute the ANT build script as follows:

```
C:\Program Files\KonaKart\custom\batch>..\bin\ant
Buildfile: build.xml

clean:
[echo] Cleanup...

compile:
[echo] Compile the batch sources
[mkdir] Created dir: C:\Program Files\KonaKart\custom\batch\classes
[javac] Compiling 4 source files to C:\Program Files\KonaKart\custom\batch\classes

make_batch_jar:
[echo] Create the batch jar
[mkdir] Created dir: C:\Program Files\KonaKart\custom\batch\jar
[jar] Building jar: C:\Program Files\KonaKart\custom\batch\jar\konakartadmin_batch.jar

build:

BUILD SUCCESSFUL
Total time: 2 seconds
```

Once rebuilt, for convenience, you can use the *copy_jars* ANT target to copy the konakartadmin_batch.jar into the konakartadmin webapp.

```
C:\Program Files\KonaKart\custom\batch>..\bin\ant
Buildfile: build.xml

copy_jars:
[echo] Copy the batch jar to the lib directory
[copy] Copying 1 file to C:\Program Files\KonaKart\webapps\konakartadmin\WEB-INF\lib

BUILD SUCCESSFUL
Total time: 0 seconds
```

After possible reconfiguration of the scheduling configuration files (which you would need to do if you have added a new batch job - see above), restart tomcat to test your new jobs.

Deletion of Expired Data

In order to keep KonaKart running efficiently, expired data should be deleted from the database at regular intervals. If not deleted, the data in some database tables, tends to grow and reduce the overall system performance. A batch job that can be scheduled through Quartz is provided for this task. The name of the job is *AdminCustomerBatchMgr.deleteTemporaryDataBatch* .

Expired sessions are deleted from the sessions table. These accumulate because most customers tend not to do a formal logout from the application after they've logged in. Another table that tends to grow is the kk_cookie table. Whenever a guest comes to the store, a cookie is created which contains the key to a kk_cookie table record. A temporary customer id is contained within this record and this id is used to insert and fetch shopping cart data so that a customer's cart is persisted even if he is a non-registered customer. The batch job deletes the cookie and cart records if they haven't been read for a programmable number of days. Finally, there is a counter table which is used to get unique ids for temporary customers. The batch job also deletes the data from this table.

Data Integrity

KonaKart includes a data integrity checking utility that can be used to check and optionally repair your data. The data integrity checker can be called directly using the KonaKart Admin API or indirectly by using one of the scripts provided.

The KonaKart data integrity checker is designed to be extended over time but currently covers the checking of Customer Groups, Order Status, Miscellaneous Items, Product, Manufacturer and Category Data Integrity only.

The KonaKart data shouldn't become corrupted under normal usage but if data is added to the KonaKart database through channels that don't preserve the integrity (such as via SQL commands) the database can lose its integrity. The purpose of the Data Integrity Checking utilities is to fix the data to restore it to a form that KonaKart can use. A common example of when the KonaKart database loses integrity is when languages are added to the system but records are not added for those languages to all KonaKart objects that must have a row for each language (for example for Order Status Names and Products).

Executing the Data Integrity Checker from a Script

The scripts are provided in the KonaKart Enterprise Extensions Edition.

Open a command shell in the utils/dataIntegrity directory that can be found immediately under your KonaKart installation directory. You can find out the arguments that are required for running the utility by entering a "?" as your argument as follows:

```
C:\Program Files\KonaKart\utils\dataIntegrity>checkDataIntegrity.bat -?

=====
Check the Integrity of the KonaKart Database
=====

Usage: DataIntegrityChecker
      -a (0|1)          - data area:
                        0 = All
                        1 = Customer Groups
                        2 = Order Statuses
                        3 = Misc Item Types
                        4 = Products
                        5 = Categories
                        6 = Manufacturers
      -u username        - username
      -p password        - password
      [-s storeId]       - storeId
      [-r]               - attempt repair - default is false
      [-e (0|1|2)]       - engine mode
      [-c]               - shared customers - default is not shared
      [-ps]              - shared products - default is not shared
      [-ae adminengine classname] - default is com.konakartadmin.bl.KKAdmin
      [-cu custom]       - custom option for DI Checker
      [-?]              - shows this usage information
```

To run the data integrity checker just to check the integrity of the data, but not attempt to fix it, run it without the "-r" parameter. For example (lines have been broken up to show the output more clearly):

```
C:\Program Files\KonaKart\utils\dataIntegrity>
  checkDataIntegrity.bat -a 1 -u admin@konakart.com -p princess -e 2
=====
Check the Integrity of the KonaKart Database
=====

11-Sep 10:48:56 WARN  (AdminDataIntegrityMgr.java:checkCustomerGroups:142)
  CustomerGroup record missing for Retail pt (id 1) language 5
```

To run the data integrity checker and attempt to fix any data integrity problems that are discovered, run it with the "-r" parameter. For example (lines have been broken up to show the output more clearly):

```
C:\Program Files\KonaKart\utils\dataIntegrity>
checkDataIntegrity.bat -a 1 -u admin@konakart.com -p princess -e 2 -r

=====
Check the Integrity of the KonaKart Database
=====
11-Sep 10:51:46 WARN  (AdminDataIntegrityMgr.java:checkCustomerGroups:142)
    CustomerGroup record missing for Retail pt (id 1) language 5
11-Sep 10:51:46 WARN  (AdminDataIntegrityMgr.java:checkCustomerGroups:155)
    CustomerGroup record added for: Retail pt (id 1) language 5
```

Configuring KonaKart to use Analytics Tools

This feature of KonaKart can be used to integrate with Google Analysts or any other analytics engines assuming their requirement is simply to insert some code into each page. Indeed, it's also possible to use this feature to insert some code into each page for any other purpose that you might have!

Configuring KonaKart to use Google Analytics

This section describes how to integrate KonaKart with Google Analytics. The integration technique is actually so generic in nature that it can be used for integrating with other analytics tools as well as Google's.

First you need to create an account with Google Analytics - see <http://www.google.com/analytics/> for details.



KonaCart - Google Analytics Integration

Setting up Google Analytics

Following Google's guidelines, set up a web site profile for your KonaCart site. By default the default page is `/konakart/Welcome.action`.

Take note of the javascript "Tracking Code" that Google generate for your profile. You will need this to configure KonaCart (see below).

Setting up KonaCart to use Google Analytics

KonaCart will insert the Google Analytics "Tracking Code" for your site into every page of the KonaCart store. For KonaCart to use your own unique code, you will have to copy this and add it to your `Messages.properties` file (or your own locale version of this file, as appropriate).

You must add the text of the "Tracking Code" on one line as the value of the `analytics.code` property. It may be convenient to use the default tracking code value that is included by default - but if you do this you must ensure that you replace the dummy UA-9999999-1 identifier with your own.

Once the analytics code is defined in the `Messages.properties` file you have to enable analytics in the KonaKart Admin App. You can find this configuration parameter under the *Configuration >> Logging* section.

Logging	
Admin App logging level	<input type="text" value="WARNING"/> <input type="checkbox"/>
KonaKart Log file Directory	<input type="text" value="C:/Progra~2/KonaKart/logs"/> <input type="checkbox"/>
Batch Log file Directory	<input type="text" value="C:/Progra~2/KonaKart/batchlogs/"/> <input type="checkbox"/>
Enable Analytics	<input type="radio"/> true <input checked="" type="radio"/> false <input type="checkbox"/>
<input type="button" value="Save"/> <input type="button" value="Cancel"/>	

Now sit back and wait for the data to be collected and enjoy studying the reports that Google produces!

Be patient, however, because data does not appear in your Google account for 24 hours or more.

Configuring KonaKart to use Other Analytics Tools

This feature of KonaKart can be used to integrate with other analytics engines assuming their requirement is simply to insert some code into each page. Indeed, it's also possible to use this feature to insert some code into each page for any other purpose that you might have!

Defining the Code to be inserted

For KonaKart to insert your own code into each page, you have to define it in your `Messages.properties` file (or your own locale version of this file, as appropriate).

You must add the code on one line as the value of the `analytics.code` property overwriting whatever is defined there by default.

Once your code is defined in the `Messages.properties` file you have to enable analytics in the KonaKart Admin App. You can find this configuration parameter under the *Configuration >> Logging* section. Whilst you may not actually be using the feature for "analytics", you still need to enable this setting so that the code is inserted in each page.

Setting up RMI Services

With the Enterprise Extensions version of KonaKart it is possible to configure KonaKart to use RMI versions of the engines. In simple terms, RMI ("Remote Method Invocation") is a way of distributing KonaKart processing over a network rather like SOAP web services but using a different protocol.

This section describes a step-by-step guide to setting up KonaKart on two servers that communicate using RMI.

Step by Step Guide to setting Up KonaKart to use RMI

This guide assumes that we will be setting up KonaKart on two machines (we'll call them chester (our "client") and wolves (our "server")).

- Install the KonaKart Enterprise Extensions on both machines. The easiest way to do this is to use the installation wizards. It is only necessary to populate the database during one of the two installations because we will be using the same database for this distributed configuration.
- On the "client" machine (chester) we will run only the KonaKart application client engine (KKAppEng) which is used by the storefront client and the Admin Application client part. The KKAppEng client engine will communicate with the application engine (KKEng) on the server machine. The Admin Application client will communicate with the Admin Engine (KKAdmin) on the server machine.

The KonaKart application server engine (KKEng) and the KonaKart admin server engine (KKAdmin) will both run on the "server" machine (wolves).

- On the client machine (chester) modify the following files:
 - webapps/konakart/WEB-INF/struts-config.xml - (Applicable only if you're using the Struts-1 storefront - this isn't required if you're using the Struts-2 storefront) ensure that only the KKAppEng plugin is started. At the bottom of this file you should set it as follows (notice how the KKEngPlugin block is commented out):

```
<!-- START: This one for a full konakart.war configuration
<plug-in className="com.konakart.plugins.KKEngPlugin">
<set-property property="propertiesPath" value="konakart.properties"/>
<set-property property="mode" value="2"/>
<set-property property="customersShared" value="false"/>
<set-property property="productsShared" value="false"/>
<set-property property="definitions-debug" value="2"/>
</plug-in>
END: This one for a full konakart.war configuration -->
<!-- START: This one for a full konakart_axis_client.war configuration -->
<plug-in className="com.konakart.plugins.KKAppEngPlugin">
<set-property property="propertiesPath" value="konakart.properties"/>
<set-property property="appPropertiesPath" value="konakart_app.properties"/>
<set-property property="mode" value="2"/>
<set-property property="customersShared" value="false"/>
<set-property property="productsShared" value="false"/>
<set-property property="definitions-debug" value="2"/>
</plug-in>
<!-- END: This one for a full konakart_axis_client.war configuration -->
```

- webapps/konakart/WEB-INF/konakart_app.properties - specify the RMI Application Engine as follows:

```
# -----
# KonaKart engine class used by the KonaKart Application users
#
# For the default engine use:          com.konakart.app.KKEng
# For the custom engine use:           com.konakart.app.KKCustomEng
# For the web services engine use:    com.konakart.app.KKWSSEng
# For the RMI services engine use:    com.konakart.rmi.KKRMIEng

#konakart.app.engineclass=com.konakart.app.KKEng
#konakart.app.engineclass=com.konakart.app.KKWSSEng
konakart.app.engineclass=com.konakart.rmi.KKRMIEng
```

- webapps/konakart/WEB-INF/konakart.properties - ensure that you specify the server machine as wolves as follows:

```
# -----
# RMI Registry Location - This is used to locate (not create) the RMI Registry
# The definition for the port that the RMI Registry will listen on is in the web.xml

konakart.rmi.host = wolves
konakart.rmi.port = 8790
```

- webapps/konakartadmin/WEB-INF/konakartadmin_gwt.properties - ensure that you specify the RMI Admin Engine as follows:

```
# -----
# KonaKart engine class used by the KonaKart Admin Application users
#
# For the default engine use:           com.konakartadmin.bl.KKAdmin
# For the custom engine use:           com.konakartadmin.app.KKAdminCustomEng
# For the web services engine use:     com.konakartadmin.ws.KKWSAdmin
# For the RMI services engine use:     com.konakartadmin.rmi.KKRMIAdminEng

#konakartadmin.gwt.engineclass=com.konakartadmin.bl.KKAdmin
#konakartadmin.gwt.engineclass=com.konakartadmin.ws.KKWSAdmin
konakartadmin.gwt.engineclass=com.konakartadmin.rmi.KKRMIAdminEng
```

- webapps/konakartadmin/WEB-INF/konakartadmin.properties - ensure that you specify the server machine as wolves as follows (in this example this is the same as the konakart.properties version above - but they don't have to be!):

```
# -----
# RMI Registry Location - This is used to locate (not create) the RMI Registry
# The definition for the port that the RMI Registry will listen on is in the web.xml

konakart.rmi.host = wolves
konakart.rmi.port = 8790
```

- On the "server" machine (wolves) there is very little to configure if you have installed using the wizard installation kits. We just need to ensure that the RMI services are configured to start up in the respective web.xml files as follows:
- webapps/konakart/WEB-INF/web.xml - enable the RMI server for the Application engine as follows (basically just ensure that this section is uncommented):

```
<!-- RMI Server -->
<servlet>
  <servlet-name>KonakartRMIServlet</servlet-name>
  <display-name>KonaKart RMI Server</display-name>
  <description>KonaKart RMI Server</description>
  <servlet-class>
    com.konakart.rmi.KKRMI Server
  </servlet-class>
  <init-param>
    <param-name>port</param-name>
    <param-value>8790</param-value>
  </init-param>
  <init-param>
    <param-name>rmiEnabled</param-name>
    <param-value>true</param-value>
  </init-param>
  <load-on-startup>20</load-on-startup>
</servlet>
<!-- End of RMI Server -->
```

- webapps/konakartadmin/WEB-INF/web.xml - similar to the above, enable the RMI server for the Admin engine as follows:

```
<!-- RMI Server -->
<servlet>
  <servlet-name>KonakartAdminRMIServlet</servlet-name>
  <display-name>KonaKartAdmin RMI Server</display-name>
  <description>KonaKartAdmin RMI Server</description>
  <servlet-class>
    com.konakartadmin.rmi.KKRMIAdminServer
  </servlet-class>
  <init-param>
    <param-name>port</param-name>
    <param-value>8790</param-value>
  </init-param>
  <init-param>
    <param-name>rmiEnabled</param-name>
    <param-value>true</param-value>
  </init-param>
  <load-on-startup>20</load-on-startup>
</servlet>
<!-- End of RMI Server -->
```

- You may wish to configure the report that is shown in the status panel of the Admin App to use the birtviewer on the wolves machine if your chester machine does not have access to the shared database. Set this in the Configuration >> Reports section of the Admin App.
- Start up tomcat on the server machine and then start up tomcat on the client machine then click on the following links (or appropriate versions in your environment!) to test:
 - Storefront Application [<http://chester:8780/konakart/>]
 - Admin App [<http://chester:8780/konakartadmin/>]

Integrating a Java Message Queue

KonaKart (Enterprise Extensions Only) provides the functionality to post messages to and read messages from java Message Queues with the integration of the Apache ActiveMQ messaging framework.

According to their web site (ActiveMQ [<http://activemq.apache.org>]) Apache ActiveMQ is the most popular and powerful open source messaging and Integration Patterns provider. ActiveMQ fully supports JMS 1.1 with support for transient, persistent, transactional and XA messaging.

Whilst ActiveMQ has a rich and extensive feature set the integration from KonaKart only accesses a small fraction of the available functionality.

The integration from KonaKart allows the programmer to post messages to a specified message queue ("postMessageToQueue") and read messages from a specified message queue ("readMessageFromQueue"). Engine API calls are provided to make these fundamental queue operations extremely easy to use. Note that these two calls provide very easy-to-use message manipulation functions but do not expose the full power of ActiveMQ. In most cases the only message queue functionality required by a KonaKart system will be to post a simple message to a queue so the simple abstraction provided is helpful in that it masks all the underlying complexity. See the javadoc on these API calls for more details.

A typical use for a message queue within a KonaKart implementation is to ensure guaranteed delivery of an order to an external system. A common approach would be to post a message containing the order information to a message queue when that order reaches a certain status in KonaKart (for example, that

status could be PAYMENT RECEIVED). In KonaKart, a convenient place to add the code to do this is in the OrderIntegrationMgr and the AdminOrderIntegrationMgr. Rather than delivering the message to the external system from these classes and risking the loss of that communication if the external system was unavailable, the order can be placed on a message queue to be picked up by the external system when it comes back on line. The delivery to the message queue is guaranteed and running in the same JVM as KonaKart (the ActiveMQ broker is "embedded" in the konakart webapp).

Examples of posting order messages to a message queue are provided in the installation kits (in the OrderIntegratorMgr, the AdminOrderIntegrationMgr and in the java API examples). An example of reading messages from the order message queue is also provided (in the java API examples).

Whilst it's conceivable that you may wish to make use of the "readMessageFromQueue" API calls to read messages sent from external systems in your IT infrastructure, there are no implementations of this API call in the standard KonaKart installation (other than example code to demonstrate how to make the calls).

Setting Up The Java Message Queue

The ActiveMQ Broker is embedded in the konakart webapp. You need to uncomment the ActiveMQ section in the konakart web.xml to enable the start-up of the broker and define a number of parameters:

```

<!-- Servlet for Apache Message Queue

Uncomment this if you want to use the Apache MQ

ApacheMQ Server parameters:
uri = The broker URI
mqEnabled = Enable (true) or Disable (false) the Apache Message Queue
mqName = A name for this Broker to make it unique
mqAdminUserName = admin username
mqAdminUserPassword = admin password
mqUserUserName = username
mqUserPassword = password
mqKonaKartQStub = users are authorised to use Queue Names starting with this prefix

-->

<!-- Apache ActiveMQ -->
<servlet>
<servlet-name>KonakartMQServlet</servlet-name>
<display-name>KonaKart MQ</display-name>
<description>KonaKart MQ</description>
<servlet-class>
  com.konakart.mq.KKMQServer
</servlet-class>
<init-param>
  <param-name>uri</param-name>
  <param-value>tcp://localhost:8791</param-value>
</init-param>
<init-param>
  <param-name>mqEnabled</param-name>
  <param-value>true</param-value>
</init-param>
<init-param>
  <param-name>mqName</param-name>
  <param-value>KonaKart.Broker.1</param-value>
</init-param>
<init-param>
  <param-name>mqAdminUserName</param-name>
  <param-value>kkadmin</param-value>
</init-param>
<init-param>
  <param-name>mqAdminUserPassword</param-name>
  <param-value>princess</param-value>
</init-param>
<init-param>
  <param-name>mqUserUserName</param-name>
  <param-value>kkuser</param-value>
</init-param>
<init-param>
  <param-name>mqUserPassword</param-name>
  <param-value>prince</param-value>
</init-param>
<init-param>
  <param-name>mqKonaKartQStub</param-name>
  <param-value>KonaKart.</param-value>
</init-param>
<load-on-startup>20</load-on-startup>
</servlet>
<!-- End of Apache ActiveMQ -->

```

In addition to the parameters in the konakart web.xml you must also define properties in the konakart.properties and konakartadmin.properties files as follows:

```
# -----
# Message Queue Configuration

konakart.mq.broker.uri      = tcp://localhost:8791
konakart.mq.username        = kkuser
konakart.mq.password         = prince
konakart.mq.orders.queue    = KonaKart.Orders.Queue
```

The "konakart.mq.broker.uri" property defines the URI of the broker.

The "konakart.mq.username" and "konakart.mq.password" properties define the credentials used to post and read messages.

The "konakart.mq.orders.queue" property defines the name of the queue to post messages to and read messages from. Note that the prefix "KonaKart." matches the "mqKonaKartQStub" parameter in the web.xml. The "mqUserUserName" defined in the web.xml (and also here in the konakart.properties file) is authorised to post messages to and read messages from queues whose name begins with this prefix.

By default the location for the broker's persistent data is defined to be "%CATALINA_HOME%/mq". This is defined in the startkonakart.bat and startkonakart.sh scripts in the KonaKart\bin directory. The location is defined as a System Property called "activemq.store.dir" and by default it is added to the CATALINA_OPTS environment variable in the above scripts.

Monitoring The Java Message Queue

Apache ActiveMQ provides a web-based monitoring tool which can be configured to monitor and administer the ActiveMQ message queue embedded within KonaKart. This tool is free and can be downloaded from the ActiveMQ [<http://activemq.apache.org>] website.

For basic monitoring tasks it's also very easy to use the jconsole tool that's provided in your java/bin directory.

Changing the standard password encryption algorithm

By default KonaKart uses the MD5 Message-Digest Algorithm to encrypt passwords. This is a one way algorithm which is used to encrypt customer passwords before they are stored in the database. During the login process when passwords are compared, the password entered by the customer is encrypted and compared with the stored encrypted password.

The class that is called to encrypt and check the password is called *Security.java* and may be found in the *KonaKart/custom/utils/src/com/konakart/util* directory. The default behaviour is for it to call the standard KonaKart encryption methods. However, if your requirements necessitate the implementation of a different encryption algorithm, the methods of *Security.java* may be customized to implement your own algorithm. Once modified the class must be compiled following the instructions in the *Programming Guide* chapter of this document.

Chapter 10. Marketing - Customer Tags and Expressions

The Enterprise version of KonaKart contains sophisticated marketing functionality that allows you to capture customer data as the customer uses your KonaKart eCommerce store; and to use that data within complex expressions in order to show dynamic content, activate promotions and send eMail communications.

What are Customer Tags?

A customer tag is an entity that can be associated with a customer (guest or logged in) and given a value for that customer. The best way of understanding what this really means is to look at the tags included in the standard KonaKart installation.

- PRODUCTS_VIEWED - A list of the most recently viewed products
- CATEGORIES_VIEWED - A list of the most recently viewed categories
- MANUFACTURERS_VIEWED - A list of the most recently viewed manufacturers
- PRODUCTS_IN_CART - A list of products in the shopping cart
- PRODUCTS_IN_WISHLIST - A list of products in the wish list
- SEARCH_STRING - The search string of the last product search made by the customer
- COUNTRY_CODE - The code of the customer's country
- CART_TOTAL - The currency total of the shopping cart
- WISHLIST_TOTAL - The currency total of the wish list
- BIRTH_DATE - When the customer was born
- LOGIN_DATE - Date of the last successful login
- IS_MALE - The sex of the customer
- PROD_PAGE_SIZE - The number of products shown on a page in list view
- ORDER_PAGE_SIZE - The number of orders shown on a page in list view
- REVIEW_PAGE_SIZE - The number of reviews shown on a page in list view

They tend to be used to keep track of a customer's actions in the storefront application (i.e. what products have been looked at, what products are in the wish list, what the customer has searched for etc.) and to store information about the customer such as whether he is male or female and what country he lives in. New customer tags can easily be added to store whatever information is important for your business requirements. If you create a new tag, for example, to keep track of how many orders the customer has placed in the last 6 months, then you also need to create a way of populating the tag value for each customer. In this example you could use a scheduled batch job that runs once a day.

Tags have a name, a description (used in the expression builder) and a type which can be:

- STRING_TYPE - The tag value is in the format of a String. i.e. To store the country from which the store is being accessed.
- INT_TYPE - The tag type is in the format of an int. i.e. To save the product id of the last product viewed by the customer.
- MULTI_INT_TYPE - The tag type allows the tag to store an array of ints. When this type is selected, the Max Number of Ints attribute determines the maximum number of ints in the array. i.e. To store the ids of the last 5 products viewed by the customer.
- DECIMAL_TYPE - The tag type is in the format of a decimal. i.e. To store the value of all items in the basket.
- DATE_TYPE - The tag type is in the format of a date. In an expression it can be used to compare the stored date with another chosen date.
- BOOLEAN_TYPE - The tag type is in the format of a boolean. i.e. To store the customer gender. The tag could be named IS_MALE and take values of true or false.
- AGE_TYPE - The tag type is used to store dates. Unlike the DATE_TYPE which can compare dates, this tag is used when in an expression you want to act on the elapsed time (or age) of an event. i.e. To store the date of the last login for the customer so you can take action if a customer hasn't logged in for two weeks. Alternatively you could store the age of a customer to offer promotions to certain age brackets.

What are Expressions?

Expressions are a way of combining customer tag values using AND/OR operators. An expression can be evaluated for a customer and always returns true or false. For example, I may want to show a banner to a customer if his cart already contains \$50.00 worth of goods and he has Product A in his wish list or he has recently viewed Product A. Since I know that the customer has shown an interest in Product A, my banner could attempt to persuade him to add it to the cart. In order to achieve this, I need to create an expression and run it for a customer before he checks out.

Expressions may be used for three different tasks:

- To show dynamic content such as a banner, as in the example above.
- To activate a promotion. The expression can be selected in the panel used to set promotion rules once a promotion has been created.
- To filter customers when sending out eMail communications. The expression can be selected in the Customer Communications panel.

Tutorial for creating an expression using the standard customer tags

Note that when creating expressions we often use the numeric internal ids of products, categories and manufacturers to identify them. These ids are visible in the admin app although they may be hidden by setting the configuration variables under the Admin App Configuration sub menu.

Display Product Ids	<input checked="" type="radio"/> true	<input type="radio"/> false
Display Manufacturer Ids	<input checked="" type="radio"/> true	<input type="radio"/> false
Display Category Ids	<input checked="" type="radio"/> true	<input type="radio"/> false

The id of a product (and SKU) may be viewed as you move your mouse over the product name in the Products panel:

Product Name	Price	Quantity	Status	Date Expected
A Bug's Life	\$35.99	10		29/10/2009
Below Id=8, SKU=123-abc-789	\$54.99	10		29/10/2009
Blade Runner - Director's Cut	\$35.99 \$30.00	17		29/10/2009
Bundle Saver	\$121.45	0		29/10/2009

It may also be viewed in the Details tab of the Edit Product panel:

Edit Product

Description Details Images Attributes Quantities Categories Merchandising Special Downloads Custom Reviews Tags

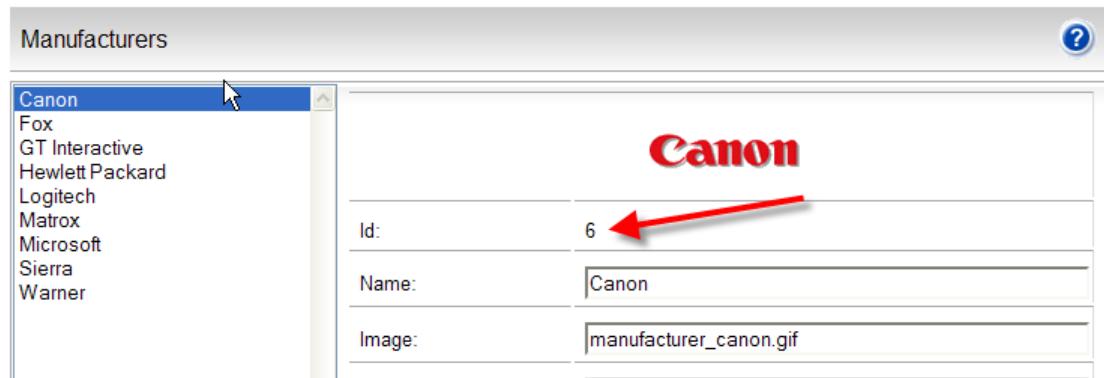
Status:	<input checked="" type="radio"/> In Stock	<input type="radio"/> Out of Stock
Id:	8	
SKU:	123-abc-789	
Product Type:	Physical Product	
Available Date:	29/10/2009	

The id of a category is visible in the Categories panel:

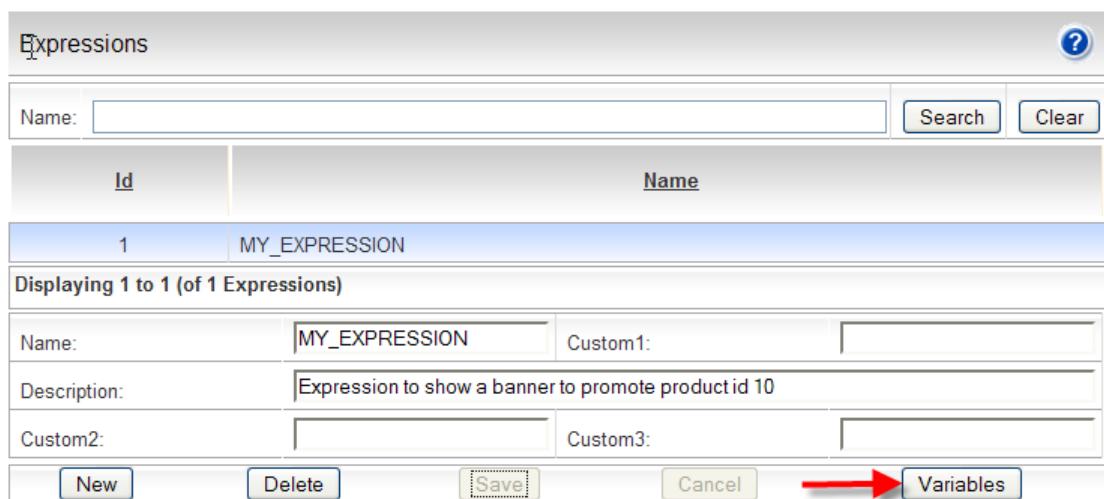
Categories

<ul style="list-style-type: none"> <input type="checkbox"/> root <input checked="" type="checkbox"/> Hardware (8) <input checked="" type="checkbox"/> Software (4) <input checked="" type="checkbox"/> DVD Movies (17) 	 <table border="1"> <tr> <td>Id:</td> <td>1</td> <td></td> </tr> <tr> <td>Category Name:</td> <td colspan="2"></td> </tr> <tr> <td>English:</td> <td colspan="2">Hardware</td> </tr> <tr> <td>Deutsch:</td> <td colspan="2">Hardware</td> </tr> </table>	Id:	1		Category Name:			English:	Hardware		Deutsch:	Hardware	
Id:	1												
Category Name:													
English:	Hardware												
Deutsch:	Hardware												

The id of a manufacturer is visible in the Manufacturers panel:



Now that we know where to find the ids, let's move on to the expression that we want to create. As explained in the example above we want to create an expression to show a banner to a customer if his cart already contains \$50.00 worth of goods and he has Product A in his wish list or he has recently viewed Product A. For the sake of the tutorial we can attribute a product id of 10 to Product A. Let's create an expression called MY_EXPRESSION as shown below.



Once the expression has been saved, we must click on the Variables button in order to open a new panel where we can enter the expression variables that define the expression.



Click the New button to create the first variable.



We choose CartTotal from the list of customer tags and give it a value of ≥ 50 . Now we must AND this tag with the the following two tags OR'ed together. To achieve this, we click the Group button.

Variables For Expression			
Cart total	\geq	50	New Grp Del
AND			New Del
Id of a product in the Wish List	$=$	10	New Grp Del
<input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Back"/>			

After entering the Customer Tag value for product in wish list, we need to OR it with the recently viewed product tag, by clicking the New button.

Variables For Expression			
Cart total	\geq	50	New Grp Del
AND			New Del
Id of a product in the Wish List	$=$	10	New Grp Del
OR	Recently viewed product id	$=$	10
<input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Back"/>			

Now we can click the save button to save the expression variables.

We have created the expression: (Cart_Total ≥ 50) AND ((prod in wishlist == 10) OR (recently viewed prod == 10))

The demo store front contains a tile called DynamicContentTile.jsp which is an example of how a tile can be displayed only if an expression evaluates to true. If you take a look at the source code of the JSP you'll see that the tile is only displayed if:

```
customerTagMgr.evaluateExpression(-1,"MY_EXPRESSION") == true
```

The CustomerTagMgr of the Client engine eventually calls the KonaKart eCommerce engine API call:

```
boolean evaluateExpression(String sessionId, int expressionId, String expressionName)
```

or

```
boolean evaluateExpressionForGuest(int customerId, int expressionId, String expressionName)
```

depending on whether the customer is logged in or not.

Note that before experimenting with expressions you must enable customer tags in the Customer Details section of the configuration variables, as shown below:

Enable Customer Tag functionality	<input checked="" type="radio"/> true	<input type="radio"/> false
Enable Customer Cart Tag functionality	<input checked="" type="radio"/> true	<input type="radio"/> false
Enable Customer WishList Tag functionality	<input checked="" type="radio"/> true	<input type="radio"/> false

The code that sets the wish list and cart customer tags is embedded within the client engine and so this tag functionality has to be enabled separately. All other customer tags are set within the Struts action classes such as CustomerRegistrationSubmitAction.java for which the source code is shipped.

At any time, you can view (and edit) the customer tag values for any customer from the Tags folder of the Edit Customer panel.

>> button points to a list of assigned tags on the right. The assigned tags table shows seven entries: CART_TOTAL (71.9800), CATEGORIES_VIEWED (:1:9:5:), PRODUCTS_IN_CART (:8:), PRODUCTS_IN_WISHLIST (:10:), PRODUCTS_VIEWED (:25:18:20:8:10:), SEARCH_STRING (siege), and WISHLIST_TOTAL (29.9900). Each entry has a 'Delete' button. At the bottom are 'Save' and 'Back' buttons."/>

CART_TOTAL	71.9800	<input type="button" value="Delete"/>
CATEGORIES_VIEWED	:1:9:5:	<input type="button" value="Delete"/>
PRODUCTS_IN_CART	:8:	<input type="button" value="Delete"/>
PRODUCTS_IN_WISHLIST	:10:	<input type="button" value="Delete"/>
PRODUCTS_VIEWED	:25:18:20:8:10:	<input type="button" value="Delete"/>
SEARCH_STRING	siege	<input type="button" value="Delete"/>
WISHLIST_TOTAL	29.9900	<input type="button" value="Delete"/>

How to set Customer Tag Values in Java code

The KonaCart Store Front Server eCommerce Engine has methods to get and set the customer tags. These methods are:

- insertCustomerTag()
- addToCustomerTag()
- getCustomerTag()
- getCustomerTagValue()

- deleteCustomerTag()
- getCustomerTags()

The CustomerTag object has getter and setter methods to set and read data as BigDecimal, Boolean, Date, Int, Int Array and String. These methods should be used rather than setting the tag value directly as a String because the internal representation of the tag data may change over time.

The best way of seeing how the customer tag data is set in the store front application is to look at the source code of the Struts Action classes where this data is set using the Client Engine. Here are a few examples:

ShowProductDetailsAction.java

```
// Set the PRODUCTS_VIEWED customer tag for this customer
kkAppEng.getCustomerTagMgr().addCustomerTag("PRODUCTS_VIEWED", selectedProd.getId());
```

QuickSearchAction.java

```
// Set the SEARCH_STRING customer tag for this customer
kkAppEng.getCustomerTagMgr().insertCustomerTag("SEARCH_STRING", searchText);
```

EditCustomerSubmitAction.java

```
// Set the BIRTH_DATE customer tag for this customer
CustomerTag ct = new CustomerTag();
ct.setValueAsDate(d);
ct.setName("BIRTH_DATE");
kkAppEng.getCustomerTagMgr().insertCustomerTag(ct);
```

Chapter 11. Reward Points

The Enterprise version of KonaKart supports reward points which enable you to increase customer loyalty and increase sales by rewarding customers for purchases as well as other actions such as registering, writing a review, referrals etc. During the checkout process, points may be redeemed for discounts up to the total value of the order.

The mechanism to redeem and allocate points is controlled by the KonaKart promotion sub-system which means that both actions may be assigned the rules applicable to all promotions. i.e. You may decide to only accept points redemption if the order value is above a limit or only allocate points for an order containing a particular product etc.

Configuration of Reward Points

In order to configure the reward point system, you must take the following steps:

The screenshot shows a configuration form titled "Reward Points". It contains three settings: "Enable Reward Points" (radio button selected for "true"), "Reward Points for registering" (text input set to 0), and "Reward Points for writing a review" (text input set to 0). At the bottom are "Save" and "Cancel" buttons.

Enable Reward Points	<input checked="" type="radio"/> true	<input type="radio"/> false
Reward Points for registering	0	
Reward Points for writing a review	0	
<input type="button" value="Save"/>		<input type="button" value="Cancel"/>

In the Configuration>>Reward Points section of the Admin App, the Reward Point functionality must be enabled. There are also configuration variables to assign a number of points for when a customer registers or writes a review. More options like this may be easily added to the solution.

The screenshot shows a table titled "Order Total Modules" with columns "Module Name" and "Sort Order". The table lists various modules with their sort orders: Shipping (2), Sub-Total (1), Tax (3), Total (40), Product Discount (red dot), Order Total Discount (red dot), Shipping Discount (red dot), Reward Points (60), and Redeem Points (30). At the bottom are "Install" and "Remove" buttons.

Module Name	Sort Order
Shipping	2
Sub-Total	1
Tax	3
Total	40
Product Discount	•
Order Total Discount	•
Shipping Discount	•
Reward Points	60
Redeem Points	30

Reward Points

The Reward Point and Redeem Point Order Total modules must be installed in the Modules>>Order Totals section of the Admin App. The sort order of these modules is important. The Redeem Points module must come before the Total module and the Reward Points module must come after. The reason for this can be explained by the image below.

Manage your Shopping Cart

Item	Quantity	Price	Total
 Nikon 1 J2 Compact	1	\$549.95	\$549.95
 Galaxy Tab 2	1	\$549.99	\$549.99
Coupon Code		Sub-Total:	\$1,099.94
Gift Certificate		Flat Rate (Shipping):	\$5.00
Redeem Reward Points (10994 points available)	1500	1500 Reward Points Redeemed:	\$15.00
		Total:	\$1,089.94
		Reward Points Earned:	10849

CHECKOUT

Red arrows point from the "Redeem Reward Points" input field to the "1500 Reward Points Redeemed:" and "Reward Points Earned:" lines in the summary table.

Now two promotions must be defined that use the modules that have just been installed:

Promotions

Name	Status	Start Date	End Date	Date Added	
No items found					
Name:	Reward Points	Promotion Type:	Reward Points		
Start Date:	Select date	End Date:	Select date		
Active:	<input checked="" type="checkbox"/>	Requires Coupon:	<input type="checkbox"/>	Cumulative:	<input type="checkbox"/>
Description:					
Minimum Order Value:	0	Min total quantity:	0		
Min quantity for a product:	0	Calculate points on amount before tax:	<input checked="" type="radio"/> true <input type="radio"/> false		
Points multiplier:	10	Include shipping cost in calculation:	<input type="radio"/> true <input checked="" type="radio"/> false		
<input type="button" value="New"/> <input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Coupons"/> <input type="button" value="Rules"/> <input type="button" value="Gift Certificates"/>					

The Reward Points Promotion is the promotion that allocates a number of points to the customer based on the value of the order. The number of points is calculated by multiplying the order value by the points multiplier attribute. i.e. If the value of the order is \$50 and the multiplier is 10, then $50 \times 10 = 500$ points will be assigned to the customer when the order is paid for.

Promotions

Name	Status	Start Date	End Date	Date Added	
No items found					
Name:	Redeem Points	Promotion Type:	Redeem Points	<input type="button" value="▼"/>	
Start Date:	Select date	End Date:	Select date		
Active:	<input checked="" type="checkbox"/>	Requires Coupon:	<input type="checkbox"/>	Cumulative:	<input type="checkbox"/>
Description:	<input type="text"/> <input type="button" value="▲"/> <input type="button" value="▼"/>				
Minimum Order Value:	0	Min total quantity:	0		
Min quantity for a product:	0	Determine min order value on amount before tax: <input checked="" type="radio"/> true <input type="radio"/> false			
Points multiplier:	0.01				
<input type="button" value="New"/> <input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Coupons"/> <input type="button" value="Rules"/> <input type="button" value="Gift Certificates"/>					

The Redeem Points Promotion is the promotion that converts points into a discount. The discount is calculated by multiplying the number of points redeemed by the points multiplier. i.e. If 1000 points are redeemed and the point multiplier is 0.01, then a discount of $1000 \times 0.01 = \$10.00$ is created. The points multiplier can also be considered to be the value of a single point. In the example above, a single point is worth 1 cent.

10994 Reward Points Available

1-3 of 3		
Date	Description	Points
04/02/2013	Points redeemed in order #1	(50)
04/02/2013	Points assigned for order #1	10994
04/02/2013	Points earned for registering	50

When a customer logs into the store front application he can view a statement of his reward point transactions and he always has available the total number of points that he can spend.

Edit Customer

Personal Address Custom Tags **Points**

Date Added	Code	Description	Points
04/02/2010	ORDER	Points redeemed in order #3	(200)
04/02/2010	ORDER	Points assigned for order #3	4880
04/02/2010	ORDER	Points assigned for order #2	300
		Total	4980

Displaying 1 to 3 (of 3 Records)

Code:	<input type="text"/>	Points:	<input type="text" value="1"/>	Add Points
Description:	<input type="text"/>			Remove Points

The administrator can view the reward point transactions for any customer, and has the ability to add and delete points from the customer's account.

Technical Details

Reward points are added and removed from a customer's account when an order is saved or changes state. The code that manages this is in the com.konakart.bl.OrderIntegrationMgr and com.konakartadmin.bl.AdminOrderIntegrationMgr.

When an order is saved but hasn't been paid for, redeemed points are reserved so that they cannot be used twice. If the order is cancelled, these reserved points are returned to the customer, otherwise they are permanently removed once the order is paid for.

If your business requires reward points to expire after a certain period, then you may run the AdminCustomerBatchMgr.removeExpiredCustomersBatch which will expire all unused points older than a specified number of days. Once the points have been expired, they are no longer available to be used by a customer.

Chapter 12. Solr Search Engine

Since version 3.0.1.0, the enterprise version of KonaKart can use the Apache Solr [<http://lucene.apache.org/solr/>] search engine. Solr gives you fully indexed search using the Jakarta Lucene search engine. As well as being very fast regardless of the size of your catalog; it caters for misspellings, synonyms, plurals and alternate spellings.

Solr can be installed on a dedicated search server or on the same server and servlet engine where KonaKart is running. KonaKart Enterprise Extensions includes a war called solr.war which will install Solr when placed in the webapps directory of Tomcat. The standard KonaKart installation already creates a solr directory where Solr can be configured and where the indexed data is stored.

Configuring KonaKart to use Solr Instructions

The first step is to install Solr. As mentioned above, this can be done on a dedicated search server or simply by dropping solr.war into the Tomcat webapps directory. The following instructions assume that Solr is installed in the same container as KonaKart.

Next, KonaKart must be told where Solr is located and also instructed to use Solr rather than the standard database search. This can be achieved in the Configuration>>Solr Search Engine panel of the Admin App as shown in the screen shot below:

Use Solr Search Server	<input checked="" type="radio"/> true	<input type="radio"/> false
Base URL of Solr Search Server	http://localhost:8780/solr	
<input type="button" value="Save"/>		<input type="button" value="Cancel"/>

Configure Solr Search Engine

Once KonaKart has been configured to use Solr, you must instruct it to index the product catalog currently in the KonaKart database. This can be done in the Tools>Manage Solr Search Engine panel of the Admin App. This tool allows you to index all of the products or to remove all of the products from Solr. The "add" operation can be performed multiple times. Products that already exist will be overwritten and not added twice.

When KonaKart is enabled to use Solr, the Solr index will be updated automatically whenever a new product is added or an existing product is edited or deleted using the Admin App.

Customization of Solr

The Solr search engine behaves differently when compared to a relational database, so we make it straightforward to customize Solr to allow you to configure the search behavior in order to satisfy your require-

ments. For example, the standard KonaKart behavior when searching for a product using a search string is to add a wild card before and after the string in order to make the search work reliably. Let's say that the name of a product is "Hewlett Packard LaserJet 1100Xi" and a search is made for Laserjet. With a relational database, the product will not be found unless it has a leading and trailing wildcard. i.e. The search string becomes %laserjet%. However, with Solr the string is tokenized and the search for laserjet returns a result without requiring any wild cards so by default they are not added because this makes the query slower and affects the behavior of synonyms. However, if a search is made for Laser, the relational database with its wild cards will return a result whereas Solr will not return a result unless a wild card is added so that the search string becomes laser*.

Under KonaKart/java_api_examples/src/com/konakart/apiexamples you will find a Java file called MySolrMgr.java with a couple of methods that allow you to define how you want wild cards to be used for Solr searches. One method is used for managing wild cards when searching for products using text searches. The other method is used for managing the wild cards when searching by matching custom fields. If you decide to change the default behavior, you must edit the konakart.properties file in order to use the new manager rather than the standard manager.

```
konakart.manager.SolrMgr = com.konakart.bl.MySolrMgr
```

When a search string contains multiple words there is an option to search for an exact match on the string or to tokenize the string into separate keywords and to search for the keywords AND'ed together. For example, rather than searching for "Electric Rotary Lawnmower" the search string becomes Electric AND Rotary AND Lawnmower. In order to activate the tokenizer you must set the *tokenizeSolrInput* attribute to true in the *ProductSearch* object which is sent as a parameter to the *SearchForProducts* API call.

By default the attributes of a product that are indexed and that can be used for searching are:

- Product Name
- Product Description including comparison data and description custom fields for all languages
- Product Model
- Product Manufacturer

In some cases you may wish to add other product attributes so that they become searchable through Solr. For example, these may include custom fields or the SKU.

Under KonaKart/java_api_examples/src/com/konakartadmin/apiexamples you will find a Java file called MySolrMgr.java with a method

```
public String getCustomSearchData(AdminProduct prod, AdminLanguage lang)
```

that overrides the method of the standard manager. It allows you to choose any data from the product object passed in as a parameter and to add it to a string (returned by the method) which will be indexed in Solr. Note that as explained in the Javadoc for the method, the indexed data will only be used by the storefront APIs when you set *whereToSearch* in the *ProductSearch* object to *ProductSearch.SEARCH_IN_PRODUCT_DESCRIPTION*.

If you decide to change the default behavior, you must edit the konakartadmin.properties file in order to use the new manager rather than the standard manager.

```
konakart.admin_manager.AdminSolrMgr = com.konakartadmin.apiexamples.MySolrMgr
```

Forcing Solr Usage

Normally Solr is only used by KonaKart when doing text searches or when returning custom facets. In order to always use Solr, for example even when returning products for a category or manufacturer, you

must set the *forceUseSolr* attribute of the *ProductSearch* object to true. The advantage in using Solr is that it will return extra information such as the minimum and maximum product prices of the result set, and manufacturer, category and price facets. The storefront application automatically uses Solr for all product search related API calls when it has been enabled.

Suggested Search

Suggested Search using Solr terms

When KonaKart is configured to use Solr (as described above), the storefront application automatically activates a suggested search widget rather than the standard search widget. As you type into the search box, a list of suggested search items appear matching the typed letters.

The KonaKart suggested search functionality uses Solr terms. For each product a number of terms are stored. The suggested search list is ordered by popularity of the term, so the more times a term has been saved, the greater chance it has of appearing in the search list. The default terms stored for each product are:

- The category name(s) of the product. i.e. Televisions
- The name of the product manufacturer. i.e. Sony
- The name of the product. i.e. Vaio
- The name of the product model. i.e. VPC-EB42FX
- The name of the category by manufacturer. i.e. Televisions by Sony, Televisions by Philips. The added word "by" is read from the admin message catalog stored as "label.by".
- The name of the manufacturer in a category. i.e. Sony in Televisions, Sony in Computers. The added word "in" is read from the admin message catalog stored as "label.in".

Each term is indexed with metadata containing the category, product and manufacturer ids so when a customer clicks on a term such as "Sony in Televisions" he is directed to a category view, displaying Sony products within the Televisions category. At this point he can choose another manufacturer remaining within the category and / or can apply extra filters such as screen size, LED, LCD etc. if the category has been set up with product tags to allow faceted search.

Under KonaKart/java_api_examples/src/com/konakartadmin/apiexamples you will find a Java file called MySolrMgr.java with a method that overrides the addTerm() method of the standard manager. This method allows you to decide which terms you want to index and which you want to exclude. By default, all terms are indexed. However, if for example all of your products belong to the same manufacturer or category, you may want to exclude certain terms. If you decide to change the default behavior, you must edit the konakartadmin.properties file in order to use the new manager rather than the standard manager.

```
konakart.admin_manager.AdminSolrMgr = com.konakartadmin.apiexamples.MySolrMgr
```

From version 6.3.0.0 the algorithm used to search for the search string within the term is configurable using regular expression. As can be seen from the image above, there are two configuration variables which contain the regex to add before the search string and the regex to add after the search string. The default for both configuration variables is "*" which means that there can be any character (.) any number of times (*) before and after the search string to ensure that it finds substrings within the term. If both of these configuration variables are left empty, then the original algorithm is used where the search string has to match from the start of the term. For example if the term is "matrox g200 MMS" and the search

string is "g200" the new algorithm will find the term whereas the old one wont. The old algorithm will only find the term with a search string of "mat...".

A Solr terms query returns all documents matching the search string including any documents that have been marked for deletion but not yet removed from the Solr index. In order for the standard Solr Commit command to remove the documents, the expungeDeletes attribute must be set to true. i.e. <commit expungeDeletes="true" />. Setting this attribute degrades the performance of the Commit operation and so it can be configured through a configuration variable (see image above). The default setting is "true" although if you are not using Suggested Search it's more efficient to set it to false. In order to completely rebuild the Solr index you may always issue the Optimize command. e.g. <http://localhost:8780/solr/update?optimize=true>.

Suggested Spelling

Functionality

The suggested spelling functionality within KonaKart is automatically used by the storefront application when a search returns no results. The API call:

```
public SuggestedSpellingItemIf[] getSuggestedSpellingItems(String sessionId, SuggestedSpellingOptionsIf options) throws KKException
```

is called with the original search string and it may return an array of SuggestedSpellingItems containing alternative spelling suggestions. The suggestions all exist in the product catalog so by clicking on the suggestion link, one or more products should be returned.

No products were found for your search of **wheeeel**. Click on any of the following suggestions to retry:

- [wheel](#)
 - [wheeled](#)
 - [wheels](#)
 - [where](#)
-

The functionality may be enabled or disabled using the Admin App under *Configuration >> Solr Search Engine*. You may also control whether spelling correction data is added to Solr for disabled products and for invisible products.

Setup

Setting up the suggested spelling functionality is not totally automatic. The file *KonaKart/solr/collection1/conf/solrconfig.xml* may need to be modified depending on the number of stores and languages supported.

A search component section and a request handler section needs to be defined for each store / language combination. Our default installation is configured for the store with a store id: "store1" and language codes: "en", "de", "es" and "pt".

```
<requestHandler name="/spell_store1_en" class="solr.SearchHandler" startup="lazy">
<lst name="defaults">
<str name="df">spell_store1_en</str>
<!-- Solr will use suggestions from both the 'default' spellchecker
     and from the 'wordbreak' spellchecker and combine them.
     collations (re-written queries) can include a combination of
     corrections from both spellcheckers --&gt;
&lt;str name="spellcheck_store1_en.dictionary"&gt;default&lt;/str&gt;
&lt;str name="spellcheck_store1_en.dictionary"&gt;wordbreak&lt;/str&gt;
&lt;str name="spellcheck_store1_en"&gt;on&lt;/str&gt;
&lt;str name="spellcheck_store1_en.extendedResults"&gt;true&lt;/str&gt;
&lt;str name="spellcheck_store1_en.count"&gt;10&lt;/str&gt;
&lt;str name="spellcheck_store1_en.alternativeTermCount"&gt;5&lt;/str&gt;
&lt;str name="spellcheck_store1_en.maxResultsForSuggest"&gt;5&lt;/str&gt;
&lt;str name="spellcheck_store1_en.collate"&gt;true&lt;/str&gt;
&lt;str name="spellcheck_store1_en.collateExtendedResults"&gt;true&lt;/str&gt;
&lt;str name="spellcheck_store1_en.maxCollationTries"&gt;10&lt;/str&gt;
&lt;str name="spellcheck_store1_en.maxCollations"&gt;5&lt;/str&gt;
&lt;/lst&gt;

&lt;arr name="last-components"&gt;
&lt;str&gt;spellcheck_store1_en&lt;/str&gt;
&lt;/arr&gt;
&lt;/requestHandler&gt;</pre>
```

The name of the request handler must follow the convention "spell_storeId_languageCode" (e.g. spell_store1_en) since the KonaKart engine will dynamically create this string in order to call the correct request handler depending on the store and language currently being used.

```
<searchComponent name="spellcheck_store1_en" class="solr.SpellCheckComponent">

    <str name="queryAnalyzerFieldType">text_general</str>

    <!-- a spellchecker built from a field of the main index -->
    <lst name="spellchecker">
        <str name="name">default</str>
        <str name="field">spell_store1_en</str>
        <str name="classname">solr.DirectSolrSpellChecker</str>
        <!-- the spellcheck distance measure used, the default is the internal
            levenshtein -->
        <str name="distanceMeasure">internal</str>
        <!-- minimum accuracy needed to be considered a valid spellcheck suggestion -->
        <float name="accuracy">0.5</float>
        <!-- the maximum #edits we consider when enumerating terms: can be 1 or 2 -->
        <int name="maxEdits">2</int>
        <!-- the minimum shared prefix when enumerating terms -->
        <int name="minPrefix">1</int>
        <!-- maximum number of inspections per result. -->
        <int name="maxInspections">5</int>
        <!-- minimum length of a query term to be considered for correction -->
        <int name="minQueryLength">4</int>
        <!-- maximum threshold of documents a query term can appear to be
            considered for correction -->
        <float name="maxQueryFrequency">0.01</float>
        <!-- uncomment this to require suggestions to occur in 1% of the documents
            <float name="thresholdTokenFrequency">.01</float>
        -->
    </lst>

    <!-- a spellchecker that can break or combine words. See "/spell" handler
        below for usage -->
    <lst name="spellchecker">
        <str name="name">wordbreak</str>
        <str name="classname">solr.WordBreakSolrSpellChecker</str>
        <str name="field">spell_store1_en</str>
        <str name="combineWords">true</str>
        <str name="breakWords">true</str>
        <int name="maxChanges">10</int>
    </lst>

</searchComponent>
```

Multiple "Spell Checkers" may be defined by each search component. In the above example, two spell checkers are defined using the *solr.DirectSolrSpellChecker* class and the *solr.WordBreakSolrSpellChecker* class.

Faceted Searching

Prices, Manufacturers and Categories

Whenever products are returned through the KonaCart API, attributes in the *ProductSearch* object can be set so that the result contains a list of Price, Manufacturer and Category facets, all of which contain information regarding the number of products available per facet. For Manufacturer and Category facets the *ProductSearch* object contains a boolean that must be set to true. In order to return price facets a *Price-*

FacetOptions object must be instantiated and attached to the *ProductSearch* object. The *PriceFacetOptions* object defines how the facets are generated by specifying the facet size, start price and end price. There is a configuration variable for the storefront application that determines whether price facets or a price slider is displayed to allow the customer to filter the search by price. The default behaviour is to display the slider.

Custom Faceted Search

KonaCart allows a configurable way to use the powerful faceted search functionality of Solr by allowing different facets to be defined for different product types. An example will be used in order to demonstrate how to use the Administration Application to configure product attributes to be used as facets.

The example uses the standard KonaCart demonstration database which contains three DVD products under the DVD Movies >> Drama category. The first step is to define a set of custom attributes which can be applied to DVDs and can be used to return facet values.

Custom Attributes	
<input type="text" value="Name:"/> <input type="button" value="Search"/> <input type="button" value="Clear"/>	
Id	Name
12	Type
11	Rating
10	Genre

Displaying 1 to 3 (of 3 custom attr descriptors)

Name:	<input type="text" value="Rating"/>	Type:	<input type="button" value="STRING_TYPE"/>	Facet:	<input type="button" value="2"/>
Set Function:	<code>option(G=General Audience,PG=Parental Guidance,R=Restricted)</code>				
Validation:					
Template:	<input type="text"/>	Msg Cat Key:	<input type="text"/>		
Custom1:					
Custom2:					
Custom3:					

As can be seen from the image, each attribute is assigned a facet number (from 1 to 30) which is used to map that attribute to a facet field in the Solr schema. In this particular example, since the custom attributes can only take a fixed number of values, a drop list is defined (in the Set Function field) containing the allowed values. Once the custom attributes have been defined, a custom attribute template must be created to group the new attributes:

Custom Attribute Templates

Name:	<input type="text"/>	Search	Clear	
<u>Id</u>	<u>Name</u>			
1	DVD Template			
Displaying 1 to 1 (of 1 custom attr templates)				
Name:	<input type="text"/> DVD Template			
Description:	<input type="text"/> Template containing custom attributes for DVDs			
Custom1:	<input type="text"/>			
Custom2:	<input type="text"/>			
Custom3:	<input type="text"/>			
<input type="button" value="New"/> <input type="button" value="Delete"/> <input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Attributes"/>				

Once the template has been inserted, the three custom attributes may be added to the template by clicking on the Attributes button. The next step is to select each of the products within the Drama category:

Products

All Prod Types	> Drama	Search SKU:			
All Products	All Manufacture	Select	Search	Clear	
<u>Product Name</u>		<u>Price</u>	<u>Quantity</u>	<u>Status</u>	<u>Date Expected</u>
Beloved		\$54.99	10	●	18/11/2011
Courage Under Fire		\$38.99 \$29.99	10	●	18/11/2011
Red Corner		\$32.00	10	●	18/11/2011
Displaying 1 to 3 (of 3 Products)					
<input type="button" value="New"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="Reviews"/> <input type="button" value="Addresses"/> <input type="button" value="Bookings"/>					

As shown below, each product within the Drama category must be associated with the DVD template created earlier.

Edit Product - Beloved

Description	Details	Images	Attributes	Quantities	Categories	Merchandising	Prices	Downloads	Custom Tags
General Details									
Id: 20 Status: <input checked="" type="radio"/> Enabled <input type="radio"/> Disabled Custom Attr Template: DVD Template <input type="button" value="Select"/>									
SKU: <input type="text"/> Product Type: Physical Product <input type="button" value="▼"/>									

Once the template has been added to the product, the values of the new custom attributes may be set by selecting allowed values from the drop lists and clicking the Save button.

Edit Product - Red Corner

Description	Details	Images	Attributes	Quantities	Categories	Merchandising	Prices	Downloads	Custom	Tags
Custom Attributes Template Attributes										
Rating	<input type="text" value="Restricted"/>									
Type	<input type="text" value="HD-DVD"/>									
Genre	<input type="text" value="Drama"/>									
<input type="button" value="Save"/> <input type="button" value="Back"/>										

At this point, we've completed the setup procedure for products. Now what needs to be done is to create three Tag Groups (one for each custom attribute) mapped to the same facet numbers as the custom attributes.

Tag Groups

Name	Description
Genre	Genre of film
Rating	DVD Rating
Type	Format of DVD

Displaying 1 to 3 (of 3 tag groups)

Language:	Rating	DVD Rating	Facet:
English:	Rating	DVD Rating	Facet: 2
Deutsch:	Rating	DVD Rating	Facet: 2
Español:	Rating	DVD Rating	Facet: 2
Português:	Rating	DVD Rating	Facet: 2

[New](#) [Delete](#) [Save](#) [Cancel](#)



As can be seen from the above image, the Rating Tag Group is mapped to facet number 2 which is the same as the Rating custom attribute. This mapping also applies to the Genre and Type Tag Groups. Once the Tag Groups have been inserted they must be associated to the Drama Category as can be seen below. The order is important this will be the order in which they are read using the KonaKart API.

The screenshot shows the 'Categories' section of the Solr Search Engine interface. On the left, a tree view lists categories under 'root': 'Hardware (8)', 'Software (4)', 'DVD Movies (17)' which includes 'Action (9)', 'Cartoons (1)', 'Comedy (2)', 'Drama (3)', 'Science Fiction (1)', and 'Thriller (1)'. The 'Drama (3)' node is selected. The main panel is titled 'Language dependent data' and has tabs for English, Deutsch, Español, and Português. The English tab is active, showing a form with 'Category Name:' set to 'Drama'. Below the name is a rich text editor toolbar with various icons. A large text area labeled 'Category Description' is empty. At the bottom of the main panel, there is a section titled 'Select Tag Groups to associate to this Category'. On the right, there is a sidebar with three sections: 'Rating', 'Type', and 'Genre'. At the bottom of the interface are buttons for Save, Cancel, New, Delete, Move, Products, and Tag Groups. A red arrow points from the text below to the 'Tag Groups' button.

The reasons for creating a Tag Group for each facet field are twofold. Within KonaKart, Solr faceted search may only be used if a category id is specified in the search query. A category id is mandatory because only similar products belonging to the same category and associated with the same template must be returned. If a DVD and another type of product such as a Keyboard were returned in the same query, it would be impossible to have a set of facets applicable to both products. When a customer clicks on a category in the storefront application, the code must retrieve the Tag Groups associated with that category and pass these to the Product Search API call. Using this information, the KonaKart engine code can determine the sort order for the returned facet data (i.e. the same as the sort order of the Tag Groups) and if Solr doesn't return any data for one or more facets, the API call still returns the facet value with no entries so that it may be displayed on the UI.

Before testing the configuration with some API calls, the products must be added to Solr as shown below:

Manage Solr Search Engine

Clicking the Add button will add all products in the catalog to the Solr search engine. Products already present will be updated.

Clicking the Remove button will remove all indexed products from the Solr search engine.

In both cases the operation may take a while and will continue asynchronously on the Solr server.

The next part of this tutorial will demonstrate how the KonaKart Engine API may be used to retrieve products in the Drama Category using queries that return facet information and how to add the facet information as a constraint.

```

/*
 * Get the tag groups for the Drama Category
 */
TagGroupIf[] groups = eng.getTagGroupsPerCategory(dramaCatId,/* getProdCount */false,
    KKConstants.DEFAULT_LANGUAGE_ID);

/*
 * Create a ProductSearch object for the search
 */
ProductSearch search = new ProductSearch();
search.setReturnCustomFacets(true);
search.setCategoryId(dramaCatId);
search.setTagGroups(groups);

ProductsIf prods = eng.searchForProducts(null, null, search, DEFAULT_LANGUAGE);

for (int i = 0; i < prods.getCustomFacets().length; i++)
{
    KKFacetIf facet = prods.getCustomFacets()[i];

    System.out.println(facet.getName() + " - " + facet.getNumber());
    if (facet.getValues() != null)
    {
        for (int j = 0; j < facet.getValues().length; j++)
        {
            NameNumberIf value = facet.getValues()[j];
            System.out.println("\t" + value.getName() + "(" + value.getNumber() + ")");
        }
    }
}

```

The above code retrieves the Tag Groups for the category. It then creates a ProductSearch object, passing it the Tag Groups, the Category Id and instructions to return custom facets. The print out from running the code can be seen below:

```

Rating - 2
G(1)
PG(1)
R(1)
Type - 3
Blu-ray(2)
HD-DVD(1)
Genre - 1
drama(3)

```

This is what we would expect because the products have been set up like this:

- Product1: Rating = PG, Type = Blu-ray, Genre = drama
- Product2: Rating = G, Type = Blu-ray, Genre = drama
- Product3: Rating = R, Type = HD-DVD, Genre = drama

We can pass a constraint to the search by adding it to the relevant Product Group as shown below. The constraint is that the rating must be "R". Note that Rating is the first Tag Group in the list.

```

/*
 * Get the tag groups for the Drama Category
 */
TagGroupIf[] groups = eng.getTagGroupsPerCategory(dramaCatId, /* getProdCount */false,
    KKConstants.DEFAULT_LANGUAGE_ID);

/*
 * Create a ProductSearch object for the search
 */
ProductSearch search = new ProductSearch();
search.setReturnCustomFacets(true);
search.setCategoryId(dramaCatId);
groups[0].setFacetConstraint("R"); // The first tag group is Rating
search.setTagGroups(groups);

ProductsIf prods = eng.searchForProducts(null, null, search, DEFAULT_LANGUAGE);

for (int i = 0; i < prods.getCustomFacets().length; i++)
{
    KKFacetIf facet = prods.getCustomFacets()[i];

    System.out.println(facet.getName() + " - " + facet.getNumber());
    if (facet.getValues() != null)
    {
        for (int j = 0; j < facet.getValues().length; j++)
        {
            NameNumberIf value = facet.getValues()[j];
            System.out.println("\t" + value.getName() + "(" + value.getNumber() + ")");
        }
    }
}
}

```

This constraints forces KonaKart to return only one product as can be seen from the print out:

```

Rating - 2
R(1)
Type - 3
HD-DVD(1)
Genre - 1
drama(1)

```

Mult-Valued Facets

In some cases it is convenient to display a facet such as color, which may have multiple values for the same product. For example a shirt may come in blue and red but not yellow. The instructions for setting up this type of facet are as follows.

- 1) For each color create a custom attribute:

Custom Attributes

<u>Id</u>	<u>Name</u>
19	Yellow
18	Blue
17	Green
16	Red
15	Camera
14	4G
13	Smartphone

Displaying 1 to 7 (of 19 custom attr descriptors) Page 1 of 3 →

Name:	Yellow	Type:	STRING_TYPE	Facet:	13
Set Function:	<code>option(yellow=yes,=no)</code>				

The Set Function should be entered as shown in the image above `option(yellow=yes,=no)` so that only one value is created to select all yellow shirts and not all non-yellow shirts. For multi-lingual web sites you should substitute the word "yellow" with a message catalog key.

- 2) For each color create a Tag Group with the same facet number as the color:

Tag Groups

Name	Description
Camera	Camera
Color	Red
Color	Green
Color	Blue
Color	Yellow
DVD Format	The format of the DVD
MPAA Movie Ratings	The MPAA rating given to each movie
Phone Options	Smartphone
Phone Options	Flash

Displaying 1 to 9 (of 9 tag groups)

English:	Color	Yellow	Facet: 13
Deutsch:	Color	Yellow	Facet: 13
Español:	Color	Yellow	Facet: 13
Português:	Color	Yellow	Facet: 13

New Delete Save Cancel

Ensure that the name of each tag group is identical as shown above where they have all been called Color (highlighted in yellow).

3) Associate the tag group to the category that contains the products.

4) Add all colors to a Custom Attribute Template and add this template to a product so that the values may be edited from the edit product panel as shown below.

Edit Product - 61 - Galaxy Tab 2 

Description Details Images Attributes Quantities Categories Merchandising Prices Downloads **Custom** Tags

Custom Attributes **Template Attributes**

Red	yes	<input type="button" value="▼"/>
Green	no	<input type="button" value="▼"/>
Blue	yes	<input type="button" value="▼"/>
Yellow	no	<input type="button" value="▼"/>

In the storefront application the color facets will be displayed as shown below:

Color

- blue (1)
- yellow (1)
- green (2)
- red (2)

The JSP that displays the facets (Facets.jsp) loops through all of the tag groups for the category but only creates a new heading if the next tag group has a different name to the previous tag group. This is why it is important for all tag groups to have the same name.

Chapter 13. Payment, Shipping and OrderTotal Modules

Module Types

KonaCart has a flexible plug-in architecture to support the addition of modules for "payment", "shipping" and "order-totals".

Payment Modules

Payment modules shipped with KonaCart are installed and configured through the Administration Application.

If you wish to add a new payment gateway and are not a programmer, please contact us. If you are a Java programmer, it is possible for you to add your own module by following the examples we provide in the package. There is a detailed guide below.

The best approach is to learn by example. We supply all of our supported modules in source code format. They can be found under the KonaCart/custom/modules/src/com/konakart/bl/modules/payment directory. All payment modules should extend com.konakart.bl.modules.payment.BasePaymentModule and implement com.konakart.bl.modules.payment.PaymentInterface.

In order to determine which Payment Modules are installed, the KonaCart engine reads the "MODULE_PAYMENT_INSTALLED" configuration property which should contain a semicolon delimited list of payment modules installed. For backwards compatibility reasons, the property may contain a list of names with a php extension (i.e. cc.php;cod.php). KonaCart ignores the extension. Let's say that you introduce a new module called "MyModule". In this case, KonaCart will look for a class called MyModule.class in the package "com.konakart.bl.modules.payment.mymodule". Note that the package name is always the class name converted to lower case.

The interface that the payment module implements is relatively simple . The main method is called getPaymentDetails(Order order, PaymentInfo info) . The module is passed in the Order object and a PaymentInfo object . The PaymentInfo object contains details on zone information so that the module can be disabled if the delivery address isn't within a zone. It also contains details used mainly for IPN (Instant Payment Notification) such as the host and port details for the return notification and a secret key that can be used to validate the return notification. The order object contains all details about the order, some of which, may be required. In the case of the PayPal example, the final piece of the puzzle is contained in a Struts action class called com.konakart.actions.ipn.PayPalAction. This action is called by PayPal in order to return the status of the payment. When received, the action class may change the state of the order as well as updating the inventory.

Admin App Payment Modules

In some cases it may be necessary to communicate with the payment gateway after the purchase has been made by the customer using the storefront application. One example is when the credit card transaction is executed only after the goods have been shipped. Another example is recurring billing.

The Admin App has an API call *callPaymentModule* which may be used for making any type of transaction on the payment gateway. When the call is made, the full class name of the payment module object is passed, along with a PaymentOptions object containing all of the required data such as the order id and credit card details (if required).

```
PaymentOptions options = new PaymentOptions();
options.setOrderId(order.getId());
options.setAction(0);
NameValuePair[] retArray = getAdminModulesMgr().callPaymentModule(
    "com.konakartadmin.modules.payment.authorize.net.AdminPayment", options);
```

An example of the AdminPayment class is provided for AuthorizeNet. It must implement the com.konakartadmin.modules.AdminPaymentIf interface and by extending com.konakartadmin.modules.AdminBasePayment it inherits useful methods for sending the data to the payment gateway. The source code of all of these classes is supplied.

Shipping Modules

Shipping modules shipped with KonaKart are installed and configured through the Administration Application.

If you wish to add a new shipping module and are not a programmer, please contact us. If you are a Java programmer, it is possible for you to add your own module by following the examples we provide in the package. The best approach is to learn by example. We supply a few examples in source code format:

- com.konakart.bl.modules.shipping.digitaldownload.DigitalDownload.java - If you sell digital download products in your store, then you should install this module.
- com.konakart.bl.modules.shipping.fedex.Fedex.java - A FedEx shipping module
- com.konakart.bl.modules.shipping.flat.Flat.java - A flat rate
- com.konakart.bl.modules.shipping.free.Free.java - You should install this module to provide free shipping.
- com.konakart.bl.modules.shipping.freeproduct.FreeProduct.java - If you have one or more products that have been configured for free shipping, then you should install this module.
- com.konakart.bl.modules.shipping.item.Item.java - A rate per item
- com.konakart.bl.modules.shipping.table.Table.java - This shipping module implements a rate per order weight or a rate based on the total cost.
- com.konakart.bl.modules.shipping.ups.Ups.java - UPS shipping module.
- com.konakart.bl.modules.shipping.zones.Zones.java - This shipping module implements a rate per item weight per zone.

In order to determine which Shipping Modules are installed, the KonaKart engine reads the "MODULE_SHIPPING_INSTALLED" configuration property which should contain a semicolon delimited list of shipping modules installed. For backwards compatibility reasons, the property may contain a list of names with a php extension (i.e. flat.php;item.php;table.php). KonaKart ignores the extension. Let's say that you introduce a new module called "MyModule". In this case, KonaKart will look for a class called MyModule.class in the package "com.konakart.bl.modules.shipping.mymodule". Note that the package name is always the class name converted to lower case.

The interface that the shipping module implements is relatively simple . The main method is called getQuote(Order order, ShippingInfo info) . The module is passed in the Order object and a ShippingInfo object . The ShippingInfo object contains details on zone information so that the module can be disabled

if the delivery address isn't within a zone. It also contains information about the number of packages and their weight etc so that the shipping cost can be calculated . The order object contains all details about the order, some of which, may be required.

An alternative approach is to contact us with details of the Shipping Module that you would like to integrate with KonaKart so that we can create the module for you.

Configuring Free Shipping

From version 2.2.3.0, free shipping can be set on a product by product basis. To set free shipping for a product, you must navigate to the Details tab of the Edit Product panel in the Admin App. There you must change the product type to "Physical Product - Free Shipping". In order for this mechanism to function correctly, you must install the Shipping Module called "FreeProduct". This module will return a shipping quote of zero if it detects that the physical products within the order all have free shipping.

Free shipping for all products can be configured in the Administration Application in the section Modules>>Order Totals by selecting the Shipping Module.

You may allow free shipping by setting the Allow Free Shipping value to "true" . In order to not show the shipping cost of zero on the order you must set Display Shipping to "false". Free shipping can be made conditional for orders over a certain amount or just for a combination of national / international orders.

The text that you display on the screen in the area that you would normally select a shipping method, is defined in WEB-INF\classes\com\konakart\bl\modules\shipping\Shipping_xx.properties. This can be set to e.g. "Free shipping for orders over \$50", "Free Shipping", "No Shipping" etc.

Note that in order to completely remove shipping from the checkout process, you will need to make modifications to the checkout process (i.e. JSPs and Struts Actions).

Configuring the Zones Shipping Module

The Zones Shipping Module allows you implement shipping rates based on the weight of the shipment, for different countries. The module can be installed and configured through the Admin App. The number of different shipping zones has to be decided before running the Admin App. It is set in the properties file:

/webapps/konakartadmin/WEB-INF/classes/com/konakartadmin/modules/shipping/zones/
Zones.properties

by editing the section:

```
# Set this to the number of zones you require.  
MODULE_SHIPPING_ZONES_NUMBER_OF_ZONES=1
```

The Admin App uses the above information to create a number of entry fields called "Zone 1 Countries", " Zone 1 Shipping Table", "Zone 1 Handling Fee", "Zone 2 Countries", Zone 2 Shipping Table" etc. depending on how many shipping zones are required.

For each shipping zone you must enter data in :

- Zone X Countries

Contains a comma separated list of two character ISO country codes that are part of a shipping zone (i.e. US, CA for USA and Canada)

- Zone X Weight Charges

Contains shipping rates to shipping zone destinations based on a group of maximum order weights. Example: 3:8.50,7:10.50,... Weights less than or equal to 3 cost 8.50 for destinations in this Zone. Weights greater than 3 but less than or equal to 7, cost 10.50 for destinations in this zone.

- Zone X Handling Fee

Handling Fee for this shipping zone

Order Total Modules

Order Total modules shipped with KonaKart are installed and configured through the Administration Application.

If you wish to add a new order total module and are not a programmer, please contact us. If you are a Java programmer, it is possible for you to add your own module by following the examples we provide in the package.

The best approach is to learn by example. We supply the source code for all of the examples that are part of an installation. These include:

- com.konakart.bl.modules.ordertotal.productdiscount.ProductDiscount.java - Discount for individual products
- com.konakart.bl.modules.ordertotal.shipping.Shipping.java - Displays the shipping cost of the order
- com.konakart.bl.modules.ordertotal.subtotal.Subtotal.java - Calculates the subtotal of the order
- com.konakart.bl.modules.ordertotal.tax.Tax.java - Calculates the tax of the order
- com.konakart.bl.modules.ordertotal.total.Total.java - Calculates the total of the order
- com.konakart.bl.modules.ordertotal.totaldiscount.TotalDiscount.java - Discount on the total of the order

In order to determine which Order Total Modules are installed, the KonaKart engine reads the "MODULE_ORDER_TOTAL_INSTALLED" configuration property which should contain a semi-colon delimited list of order total modules installed. For backwards compatibility reasons, the property may contain a list of names with a php extension (i.e. ot_subtotal.php;ot_tax.php;ot_shipping.php). KonaKart ignores the extension. Let's say that you introduce a new module called "MyModule". In this case, KonaKart will look for a class called MyModule.class in the package "com.konakart.bl.modules.ordertotal.mymodule". Note that the package name is always the class name converted to lower case.

The interface that the order total module implements is relatively simple . The main method is called getOrderTotal(Order order, boolean dispPriceWithTax, Locale locale) . The module is passed in the Order object which contains all details about the order, some of which, may be required.

An alternative approach is to contact us with details of the Order Total Module that you would like to integrate with KonaKart so that we can create the module for you.

How to Create a Payment Module

This section explains how to create a new payment module for KonaKart using its module plug-in framework.

Introduction

KonaKart supports additional payment/shipping/order total/discount modules using a simple plug-in model.

Payment modules are typically designed to interface between KonaKart and a 3rd Party payment gateway supplier - like PayPal, Authorize.Net or WorldPay etc.

You can install as many payment gateways as you like with KonaKart but typically you would just have one configured as there is usually a cost associated with the merchant account that you have to set-up with each payment organization in order to receive your payments.

Suppose you want to create a brand new payment gateway module to interface between KonaKart and your chosen payment gateway supplier. For the purposes of this guide, let's call the payment gateway supplier "KonaPay" (a completely fictitious gateway).

Study the "KonaPay" APIs

First of all you should find out as much as you can about which interfaces "KonaPay" supports. They might supply an XML interface, a HTTP post interface, a SOAP interface or any kind of weird custom API that allows you to communicate with them. Some payment gateways do not provide any APIs for providing credit card information at all and require that your user is directed to their site to enter such sensitive details.

Choose which Interface Type you want for your users

So, you have to consider which type of gateway you want to implement for your users? This may well affect which gateway supplier you choose. Do you want to collect the credit card details yourself within KonaKart or do you want to send the users off to a payment gateway site to collect this information? Some prefer the former, some the latter.

You will see in the code that you need to set the paymentType on the PaymentDetails object, which defines which of these two modes you want:

```
// For Authorize.Net, YourPay etc.  
setPaymentType(PaymentDetails.SERVER_PAYMENT_GATEWAY);  
  
or  
  
// For PayPal, WorldPay etc.  
setPaymentType(PaymentDetails.BROWSER_PAYMENT_GATEWAY);
```

For the sake of our discussion, KonaPay provides a well-documented HTTP API that allows us to collect credit card information on the KonaKart site and send these off synchronously for payment authorization - which we think provides the best user experience at payment time. It's attractive to us because we can retain the same look and feel of our KonaKart store across the full shopping experience - rather than having to jump off to a payment gateway page provided by the gateway supplier which doesn't look anything like the rest of our site.

OK, we've chosen "KonaPay" because it fits our preferred implementation model and it has a well-documented API, what next?

Sign up for a Test Account with "KonaPay"

Typically, the payment gateway supplier will provide a free dummy test account so the next step is to apply for one of these at "KonaPay". Fortunately, "KonaPay" do provide a test account so we sign up and study the API documentation in a little more detail.

Determine which of the existing payment modules is the closest match

The next important step is to study the existing payment modules to see which one looks to be the closest fit to "KonaPay". This step is very important and will save you a great deal of time if you pick a close match.

The good news is that many of the gateways work in very similar ways within the two distinct payment module groups:

- Payment details are collected inside KonaKart ("Server Mode" - Authorize.Net, YourPay, PayJunction, USAePay)
- Payment details are collected at Gateway Supplier's Site ("Browser Mode" - WorldPay, PayPal, Chrono-Pay, ePayBg)

Copy the files of the closest match as the starting point

We decide that PayJunction is the closest match to "KonaPay" so we make copies of all the PayJunction files and directories - ensuring that we are completely consistent with the naming conventions used by PayJunction (ie, the same lower/mixed case conventions as PayJunction). Make copies of the various properties files as well and rename these as appropriate to "KonaPay", "Konapay" or "konapay" in a consistent manner.

There will be payment directories to copy under com.konakart.bl.modules.payment and com.konakartadmin.modules.payment.

We also need to copy modules/src/com/konakart/actions/gateways/PayjunctionAction.java to modules/src/com/konakart/actions/gateways/KonapayAction.java

Define the configuration parameters

Next, we need to decide which configuration options we will allow an administrator to change once the "KonaPay" module is installed.

PayJunction allows these to be configured: (the following is the code that initializes these in modules/src/com/konakartadmin/modules/payment/payjunction/Payjunction.java:)

```

configs[i++] = new KKConfiguration(
    /* title */"Enable PayJunction Module",
    /* key */"MODULE_PAYMENT_PAYJUNCTION_STATUS",
    /* value */"true",
    /* description */ "Do you want to accept PayJunction payments? ('true' or 'false')",
    /* groupId */groupId, /* sort Order */i, /* useFun */"",
    /* setFun */"choice('true', 'false')",
    /* dateAdd */now);

configs[i++] = new KKConfiguration(
    /* title */"Sort order of display.",
    /* key */"MODULE_PAYMENT_PAYJUNCTION_SORT_ORDER",
    /* value */"0",
    /* description */ "Sort order of display. Lowest is displayed first.",
    /* groupId */groupId, /* sort Order */i, /* useFun */"", /* setFun */"", /* dateAdd */now);

configs[i++] = new KKConfiguration(
    /* title */"Payment Zone",
    /* key */"MODULE_PAYMENT_PAYJUNCTION_ZONE",
    /* value */"0",
    /* description */ "If a zone is selected, only enable this payment method for that zone.",
    /* groupId */groupId, /* sort Order */i,
    /* useFun */"tep_get_zone_class_title",
    /* setFun */"tep_cfg_pull_down_zone_classes",
    /* dateAdd */now);

configs[i++] = new KKConfiguration(
    /* title */"PayJunction Username",
    /* key */"MODULE_PAYMENT_PAYJUNCTION_USERNAME",
    /* value */"pj-ql-01",
    /* description */ "The username used to access the PayJunction service",
    /* groupId */groupId, /* sort Order */i, /* useFun */"", /* setFun */"", /* dateAdd */now);

configs[i++] = new KKConfiguration(
    /* title */"PayJunction Password",
    /* key */"MODULE_PAYMENT_PAYJUNCTION_PASSWORD",
    /* value */"pj-ql-01p",
    /* description */ "The password used to access the PayJunction service",
    /* groupId */groupId, /* sort Order */i, /* useFun */"", /* setFun */"", /* dateAdd */now);

configs[i++] = new KKConfiguration(
    /* title */"Payment Server URL",
    /* key */"MODULE_PAYMENT_PAYJUNCTION_URL",
    /* value */"https://payjunction.com/quick_link",
    /* description */ "URL used by KonaKart to send the transaction details",
    /* groupId */groupId, /* sort Order */i, /* useFun */"", /* setFun */"", /* dateAdd */now);

configs[i++] = new KKConfiguration(
    /* title */"Security Options",
    /* key */"MODULE_PAYMENT_PAYJUNCTION_SECURITY",
    /* value */"AWZ|M|false|true|false",
    /* description */ "Security Options for Pay Junction - refer to PayJunction documentation for details",
    /* groupId */groupId, /* sort Order */i, /* useFun */"", /* setFun */"", /* dateAdd */now);

configs[i++] = new KKConfiguration(
    /* title */"Debug Mode",
    /* key */"MODULE_PAYMENT_PAYJUNCTION_DEBUG_MODE",
    /* value */"true",
    /* description */ "If set to true, the PayJunction module will be set to debug code",
    /* groupId */groupId, /* sort Order */i, /* useFun */"",
    /* setFun */"choice('true', 'false')",
    /* dateAdd */now);

```

The above are quite a typical combination of fields. Let's look at each one in turn:

- **MODULE_PAYMENT_PAYJUNCTION_STATUS** - for enabling/disabling the module. Typically provided for all modules.
- **MODULE_PAYMENT_PAYJUNCTION_SORT_ORDER** - for order the module is displayed if there are more than one available. Typically provided for all modules

- MODULE_PAYMENT_PAYJUNCTION_ZONE - for defining a zone where this payment module can be used. Typically provided for all modules.
- MODULE_PAYMENT_PAYJUNCTION_USERNAME - username to access the payment gateway service. Often required for payment modules
- MODULE_PAYMENT_PAYJUNCTION_PASSWORD - password to access the payment gateway service. Often required for payment modules
- MODULE_PAYMENT_PAYJUNCTION_URL - the payment service URL. Often required for payment modules (useful for switching between the payment service's test and live services).
- MODULE_PAYMENT_PAYJUNCTION_SECURITY - a special configuration option particular to PayJunction. Typically payment modules would have one or two of these, but they would be different between gateway services.
- MODULE_PAYMENT_PAYJUNCTION_DEBUG_MODE - handy for diagnosing problems, especially in development. Typically provided for all modules

For "KonaPay" we will change the configuration parameter names to include "KONAPAY" rather than "PAYJUNCTION", so we would have "MODULE_PAYMENT_KONAPAY_STATUS" etc... We need to replace _PAYJUNCTION_ with _KONAPAY_ throughout all the source and properties files. For "KonaPay" we will assume, for simplicity that we will define the same set as is defined for PayJunction except the MODULE_PAYMENT_PAYJUNCTION_SECURITY variable which we'll exclude.

Understanding the Configuration Options

Looking more closely at the KKConfiguration structure that is initialized for each configuration object:

```
configs[i++] = new KKConfiguration(
    /* title */"Enable PayJunction Module",
    /* key */"MODULE_PAYMENT_PAYJUNCTION_STATUS",
    /* value */"true",
    /* description */"Do you want to accept PayJunction payments? ('true' or 'false')",
    /* groupId */groupId,
    /* sort Order */i,
    /* useFun */"",
    /* setFun */"choice('true', 'false')",
    /* dateAdd */now);
```

The comments should help a lot in understanding what these attributes are for. In more detail:

- title - affects what's shown on the screen of the Admin App
- key - the unique key by which this key is known
- value - the default value
- description - this is provided as floatover help in the Admin App
- groupId - the groupId - this is 6 for payment modules
- sortOrder - the order in which these keys are displayed in the Admin App
- useFun - a function that defines how this key should be "used"

- setFun - a function that defines how this key is "set"
- dateAdd - the date the key is added to the database.

The only slightly complicated ones are the use and set functions. The example above shows the setFun to use to display a true/false toggle in the Admin App. The setFun and useFun can be null for most simple text fields. These setFun and useFun formats are used throughout the Admin App and perhaps the easiest way to see how they work is to look at these columns in the database for each configuration key that uses them.. eg: Issue "SELECT configuration_title, configuration_key, use_function, set_function FROM configuration;" against your database to see lots of examples.

** Ask questions in the forum if you're uncertain about these.

Add the "KonaPay" gateway to the Admin App

In order to see the "KonaPay" payment gateway module in the Admin App we have to add it to the list of payment modules in webapps/konakartadmin/WEB-INF/classes/konakartadmin.properties. Remember to match the classname - so be careful with your lower case/upper case characters! Once set, the Admin App will include the "KonaPay" module in the set of modules that can be installed or removed from the system: (Konapay has been added in bold below):

```
# -----
# Set the class names of the various modules you would like to make
# available. The administrator can still choose to enable or disable
# these.
#
# Note that if you remove a module from the definitions below that
# has already been set up in the database the users may still have
# access to the modules in the konakart application. Hence, it is
# advisable to remove the modules before they are removed from these
# definitions.

# Make these space or semi-colon-separated class names - they have
# implied prefixes of:
#
#     com.konakartadmin.modules.payment.{lower case module name}.
#     com.konakartadmin.modules.shipping.{lower case module name}.
#     com.konakartadmin.modules.orderTotal.{lower case module name}.

konakart.modules.payment=Authorizenet Yourpay Payjunction Konapay
konakart.modules.shipping=Flat Item Table Zones Free DigitalDownload
konakart.modules.ordertotal=Tax Total ProductDiscount TotalDiscount
```

Implement the PaymentInterface

Next we need to implement the PaymentInterface in our Konapay class in the com.konakart.bl.modules.payment.konapay package. If the PayJunction directories were copied correctly earlier the java file should be at custom/modules/src/com/konakart/bl/modules/payment/konapay

This Konapay.java source extends BasePaymentModule (whose source is provided at custom/appn/src/com/konakart/bl/modules/payment/BasePaymentModule.java) and implements PaymentInterface.

It's advisable to follow the pattern you see in the file you copied. Therefore, rename all the constants and variables to names appropriate for "KonaPay" and set up the PaymentDetails object in a similar way in getPaymentDetails().

For much of the PaymentDetails, the values set will be very similar between payment modules (eg. the payment module code, the sort order, title and description etc) but there will always be differences in the "NameValuePair[]" parameters that are added further down in the getPaymentDetails() method.

NameValue[] Parameters

Each payment gateway will need different parameter names so to work out which NameValue pairs you need to add you need to study the API specification. With PayJunction, you can see that the following are required: dc_logon, dc_password, dc_transaction_type etc. With "KonaPay" there will be different names and perhaps different encoding rules; the API documentation will define the requirements. Check the examples in the gateway documentation, these often make it clear what the names and values should be. Once the parameters are understood, add these to the PaymentDetails - they will be used later to build the request to the gateway supplier.

Implement the Action code

Next we need to implement the Action code in our KonapayAction class in the com.konakart.actions.gateway package. The "KonaPay" java file should be called custom/modules/src/com/konakart/actions/gateways/KonapayAction.java. (Notice that the (usually synchronous) modules which collect the payment information from the user on the KonaCart site have to have "com.konakart.actions.gateway" implementations, whereas the (asynchronous) modules that divert the user off to a page on the gateway's site have to have "com.konakart.actions.ipn" implementations).

This Konapay.java source extends BasePaymentModule (whose source is provided at custom/appn/src/com/konakart/bl/modules/payment/BasePaymentModule.java) and implements PaymentInterface.

It's advisable to follow the pattern in the file you copied from PayJunction. Therefore, rename all the constants and variables to names appropriate for "KonaPay" and set up the PaymentDetails object in a similar way in getPaymentDetails().

The Action files are quite well documented so it should be easy to see where to make modifications for a new module. Basically you will have to create your message to post to the gateway then process the returned information to determine success or failure. If there's a failure you can return the user to the credit card screen to try again (and show the error message that you received). Alternatively, for more serious errors, you can show the exception in a standard form for the KonaCart site.

Save IPN details

In order for your KonaCart administrator to diagnose payment gateway problems in the future, it's a good idea to save details of each transaction with a call to "saveIpnHistory" (see kkAppEng.getEng().saveIpnHistory()). Set the various attributes on IpnHistoryIf to values that you derive from the response as you see in the example source. These records are available in the KonaCart Admin App for inspection and can be useful when diagnosing problems with payment gateways. The records can also be useful in recording unique transaction codes that are provided by the gateway suppliers for proving that payments have been made, should that need ever arise.

Save the gateway response to a file

Another technique that you can use to diagnose problems is to save gateway responses to a file on disk. An example of such code is in the YourpayAction source as follows:

```
if (yourPayDebugMode)
{
    // Write the response to an HTML file which is handy for
    // diagnosing problems

    try
    {
        String outputFilename = getLogFileDirectory()
            + "yourpay_resp_"
            + order.getId()
            + ".html";
        File myOutFile = new File(outputFilename);
        if (log.isDebugEnabled())
        {
            log.debug("Write gateway response to " + myOutFile.getAbsolutePath());
        }
        BufferedWriter bw = new BufferedWriter(new FileWriter(myOutFile));
        bw.write(gatewayResp);
        bw.close();
    } catch (Exception e)
    {
        // dump the exception and continue
        e.printStackTrace();
    }
}
```

Note that the yourPayDebugMode boolean could be set by one of your module configuration parameters and the log file location that is returned by the getLogFileDirectory() call is defined in the Admin Application under the Configuration >> Logging section.

Send payment confirmation email

You have a choice whether or not to send payment confirmation mails in the Action class. (You are also free to change the format of these emails if you wish by modifying the velocity templates). The call for sending emails is simply kkAppEng.getEng().sendOrderConfirmationEmail().

Struts mapping

This tutorial was written for a Struts-1 MVC implementation but the concepts are the same for Struts-2. The easiest way to understand what you have to do is to copy the Actions and struts.xml section for the payment gateway you are using as a template for KonaPay.

More often than not the Struts mapping code in the Action class will be the same for all the synchronous (or "server-side") payment modules. Each "mapping string" in the mapping.forward("mapping string") calls must match the struts-config.xml file as follows: The entry for "KonaPay" has been added below:

```
<!-- ===== Server Side Gateways -->
<action path="/USAePay"
    type="com.konakart.actions.gateways.UsaepayAction">
    <forward name="Approved" path="/CheckoutFinished.do"/>
    <forward name="TryAgain" path="/CheckoutServerPayment.do"/>
    <forward name="NotLoggedIn" path="/CheckoutDelivery.do"/>
</action>

<action path="/AuthorizeNet"
    type="com.konakart.actions.gateways.AuthorizenetAction">
    <forward name="Approved" path="/CheckoutFinished.do"/>
    <forward name="TryAgain" path="/CheckoutServerPayment.do"/>
    <forward name="NotLoggedIn" path="/CheckoutDelivery.do"/>
</action>

<action path="/PayJunction"
    type="com.konakart.actions.gateways.PayjunctionAction">
    <forward name="Approved" path="/CheckoutFinished.do"/>
    <forward name="TryAgain" path="/CheckoutServerPayment.do"/>
    <forward name="NotLoggedIn" path="/CheckoutDelivery.do"/>
</action>

<action path="/KonaPay"
    type="com.konakart.actions.gateways.KonapayAction">
    <forward name="Approved" path="/CheckoutFinished.do"/>
    <forward name="TryAgain" path="/CheckoutServerPayment.do"/>
    <forward name="NotLoggedIn" path="/CheckoutDelivery.do"/>
</action>

<action path="/YourPay"
    type="com.konakart.actions.gateways.YourpayAction">
    <forward name="Approved" path="/CheckoutFinished.do"/>
    <forward name="TryAgain" path="/CheckoutServerPayment.do"/>
    <forward name="NotLoggedIn" path="/CheckoutDelivery.do"/>
</action>
```

One other change is required in struts-config.xml (Struts-1) which defines the forwarding for the server-side payments: It's critical to maintain naming consistency here (use lower or mixed case as in the other examples - or whatever it takes to match your definitions elsewhere).

```
<action path="/CheckoutServerPaymentSubmit"
    type="com.konakart.actions.CheckoutServerPaymentSubmitAction"
    name="CreditCardForm" scope="request" validate="true"
    input="/CheckoutServerPayment.do">
    <forward name="usaepay" path="/USAePay.do"/>
    <forward name="authorizenet" path="/AuthorizeNet.do"/>
    <forward name="payjunction" path="/PayJunction.do"/>
    <forward name="konapay" path="/KonaPay.do"/>
    <forward name="yourpay" path="/YourPay.do"/>
</action>
```

Build, Deploy and Test

Once the java code is built, the properties files and the struts-config.xml have been updated, all that is required to be done is to execute the ant build, copy the new jars into position, restart tomcat and test!

The gateway suppliers typically provide test credit card numbers for you to use for these tests.

If you have questions on customizing KonaCart please post these on our forum [<http://www.konakart.com/forum>] , at <http://www.konakart.com/forum>

How to Create a Promotion Module

This section explains how to create a new Promotion module for KonaKart using its module plug-in framework. The process for creating a new promotion module is similar to that for creating a new payment module described above so here we'll concentrate mainly on how to define the data entry configuration widgets for a promotion module.

Determine which of the existing Promotion modules is the closest match

The first important step is to study the existing Promotion modules to see which one looks to be the closest fit to the type of promotion you are considering. This step is very important and will save you a great deal of time if you pick a close match. For example, TotalDiscount.java provides a percentage or amount discount on the total of the order and ProductDiscount.java provides a percentage or amount discount for individual products. These could be good starting points for your custom promotion.

Copy the files of the closest match as the starting point

If you decide that the TotalDiscount promotion is a good starting point then the next step is to copy the following promotion directories and contents:

- modules/src/com/konakart/bl/modules/ordertotal/totaldiscount
 - modules/src/com/konakartadmin/modules/ordertotal/totaldiscount
- to
- modules/src/com/konakart/bl/modules/ordertotal/mytotaldiscount
 - modules/src/com/konakartadmin/modules/ordertotal/mytotaldiscount

Rename the files from TotalDiscount*.* to MyTotalDiscount*.*

MyTotalDiscount.java under the konakart directory contains the code that runs in the storefront and calculates the promotion discount. MyTotalDiscount.java under the konakartadmin directory is used to define the parameters for configuring the promotion.

Define the configuration parameters

Next, we need to decide which configuration options we will allow an administrator to change when creating a new promotion. New promotions are created from the Admin App under Marketing >> Promotions. You pick a promotion type from the drop list and a number of entry fields appear into which you can enter data that configures the promotion, such as the actual discount amount or percentage value.

The configuration parameters are defined in the file modules/src/com/konakartadmin/modules/ordertotal/mytotaldiscount/MyTotalDiscount.java . Each configuration attribute must have a unique key. The first two attributes are similar for all promotion modules since they define whether the module is active and the sort order when displayed in a list in the Admin App under Modules >> Order Totals.

```
configs[i] = new KKConfiguration(
    /* title */"Minimum Order Value",
    /* key */"MODULE_ORDER_TOTAL_TOTAL_DISCOUNT_MIN_ORDER_VALUE",
    /* value */"custom1",
    /* description */"The discount only applies if the total of the order,"
        + " equals or is greater than this minimum value",
    /* groupId */groupId,
    /* sortO */i++,
    /* useFun */"invisible",
    /* setFun */"double(0,null)",
    /* dateAdd */now);
```

Let's look at the example above in order to understand the configuration details.

- title - The label for the data entry widget when configuring the promotion. If an entry in the format cfg.title.key exists in the message catalog then the label is retrieved from the catalog. e.g. cfg.title.MODULE_ORDER_TOTAL_TOTAL_DISCOUNT_APPLY_BEFORE_TAX = Apply before tax.
- key - The key must be unique. All of our modules tend to follow a naming convention as you can see in the example above.
- value - Each promotion can have up to 10 configuration parameters. The value must be in the range "custom1" to "custom10".
- description - The description is what a user sees in the Admin App as floatover help when the mouse is over the label of the data entry widget. If an entry in the format cfg.desc.key exists in the message catalog then the description is retrieved from the catalog. e.g. cfg.desc.MODULE_ORDER_TOTAL_TOTAL_DISCOUNT_APPLY_BEFORE_TAX = If set to true, the discount is calculated on the amount before tax.
- groupId - Set to 6 for promotion modules.
- sortO - This should be left as i++ so that the data entry widgets in the UI match the sort order in which they are defined in the file.
- useFun - Set to "invisible" for promotions so that they don't appear when activating the module.
- setFun - This is an important attribute since it defines what the data entry widgets will look like and allows you to add validation. Valid values are:
 - integer(min,max) where min and max may be numbers or set to null. i.e. integer(10,null) checks that the integer has a minimum value of 10 or above. Integer(null,null) just checks that the value entered is an integer without doing any range checking.
 - double(min,max) using the same logic as explained above for decimal numbers.
 - string(min,max) using the same logic as explained above for the length of the string.
 - choice('true','false') creates a radio button panel with the options of true and false. choice('msg.small','msg.medium','msg.large') creates 3 radio buttons where the labels are looked up in a message catalog. In this case, the text displayed is looked up from the message catalog, whereas the values saved are always msg.small, msg.medium and msg.large regardless of the language.
 - option(0=date.day.long.Sunday,1=date.day.long.Monday,2=date.day.long.Tuesday) creates a drop list for the first 3 days of the week. The text of the days displayed is retrieved from the message catalog whereas the saved data is 0 for Sunday, 1 for Monday etc.

- RichText(x) or RichText(x,min,max) where x defines the vertical size (in elements) of the Rich Text data entry widget. (e.g. RichText(10) for a size of 10em). min and max provide validation for the length of the string. e.g. If a value is mandatory then min should be set to a value greater than zero. This widget spans a complete row rather than half a row like the other widgets.
- dateAdd - Set to now.

The screenshot shows a configuration interface with several input fields and a rich text editor:

- Top Row:** Two input fields: "double(0,null):" with value "4.77" and "integer(0,null):" with value "56".
- Middle Row:** Two input fields: "text(1,128):" with value "ABC" and "choice('true', 'false'):" with radio buttons for "true" (selected) and "false".
- RichText Editor:** A large text area containing "Hello". Below it is a label "RichText(10,0,20)". Above the text area is a toolbar with various rich text buttons.
- Bottom Left:** A dropdown menu labeled "option(" with code: "0=date.day.long.Sunday, 1=date.day.long.Monday, 2=date.day.long.Tuesday"). The dropdown menu shows items: "Sunday" (selected), "Monday", and "Tuesday".
- Buttons:** A row of buttons: New, Delete, Save, Cancel, Coupons, Rules, and Gift Certificates.

Promotion Configuration Widgets

The above image shows the widgets that are created based on the setFun parameter. In order to display the setFun value being used, it has been added to the label and highlighted in yellow.

Add the new promotion module to the Admin App

In order to see the new promotion module in the Admin App under Modules >> Order Totals we have to add it to the list of Order Total modules in webapps/konakartadmin/WEB-INF/classes/konakartadmin.properties. Remember to match the classname - so be careful with your lower case/upper case characters! Once set, the Admin App will include the new module in the set of modules that can be installed or removed from the system.

Build, Deploy and Test

Once the java code is built and the properties file has been updated, all that is required to be done is to execute the ant build, copy the new jars into position, restart tomcat and test!

Chapter 14. Recurring Billing

KonaKart provides support for recurring billing which may be implemented in one of two alternate ways. The actual payment transactions may be performed by a payment gateway that supports recurring billing, or they may be performed by a KonaKart batch program that interfaces with the payment gateway to make the regular payments. We recommend the first approach since it doesn't require that you store sensitive credit card information within your local database. The credit card details are transmitted to the payment gateway so that it can create the subscription and automatically make the payments at predefined intervals.

There are two main objects within KonaKart that provide recurring billing support. These are the Payment Schedule object and the Subscription object. Both have dedicated panels within the Admin App so that they may be managed. The Payment Schedule panel may be found under Products, whereas the Subscription Panel may be found under Orders.

Payment Schedule

A payment schedule defines the frequency of payments for a product as well as the total number. i.e. Payments may be monthly, lasting for two years. In the case of monthly payments, it allows you to define the day of the month the payment should be made. It also allows you to define one or more trial payments which may be for a different amount. i.e. The subscription may consist of monthly payments of \$9.99 for the first 6 months before increasing to \$30.00 .

A payment schedule is associated to a product. This is done in the Edit Product panel. A drop list containing payment schedules will appear in the Details tab of the Edit Product panel if one or more payment schedules exist. When an order is confirmed by a customer during the checkout process, the products within the order are examined to detect whether any of them have an associated payment schedule. If a payment schedule is found, this triggers off the process of creating a recurring billing subscription.

Subscription

A subscription defines the payment amount and start date for the payments. It can be activated / deactivated and if any problems are found (i.e. credit card expired) during the payment transaction, these problems are logged in the subscription object . As mentioned above, a subscription is created and saved at the moment when an order is confirmed, if any of the products within the order are associated to payment schedules. A subscription can be associated with one or more Payment Info objects, each of which records the amount paid on a particular date, as well as saving the complete transaction reply from the payment gateway.

A subscription object contains many other attributes such as the last and next billing dates, the order number, the product SKU and custom fields. These are all optional, but may be used to save important information that can be used for reporting purposes and to control the actual payments when they are being managed by KonaKart.

Using a Payment Gateway that supports Recurring Billing

Many payment gateways support recurring billing and provide interfaces to create and manage subscriptions. KonaKart includes a recurring billing implementation for AuthorizeNet in the file called `AdminPayment.java` in the `KonaKart\custom\modules\src\com\konakartadmin\modules\payment\authorizeNet` directory. This file contains methods to create, update and cancel an AuthorizeNet subscription

as well as a method to read the status of a subscription in order to determine whether it is active or has been cancelled etc. The file also contains a method to perform a standard credit card payment through AuthorizeNet which could be used when the recurring billing is managed by KonaKart rather than the payment gateway. In order to create something similar for other payment gateways, this file should be copied into the relevant directory (e.g. KonaKart\custom\modules\src\com\konakartadmin\modules\payment\myGateway) and customized in order to implement the protocol of the chosen gateway.

The way to call these payment gateway methods is through the Admin Engine which contains a method to call a payment gateway called:

```
public NameValue[] callPaymentModule(String sessionId, String moduleClassName, PaymentOptions options) throws KKAdminException;
```

The PaymentOptions object contains data to configure the method and the moduleClassName is the actual name of the class that is instantiated.

In order to manage a recurring billing subscription the steps are:

Insert a subscription

A subscription must be inserted into the KonaKart database and also sent to the payment gateway since it is the payment gateway that will actually perform the billing. The trigger for this operation is when a customer confirms an order in the storefront application. The storefront code must create a subscription which is saved in the KonaKart database. An example can be found in the file *AuthorizenetAction.java* under the directory KonaKart\custom\modules\src\com\konakart\actions\gateways. The method (which is commented out) is called *manageRecurringBilling()* . A Subscription object is created and inserted into the KonaKart database using the *insertSubscription()* API call . In order to be able to also create a subscription through the payment gateway you can add code to the *OrderIntegrationMgr* which can be found in the directory KonaKart\custom\appn\src\com\konakart\bl . As you can see, the OrderIntegrationMgr has methods that are called before and after inserting and updating a subscription. The example code can be found in the *afterInsertSubscription()* method.

```
/*
 * Create a PaymentOptions object to pass to the method that calls the payment gateway.
 * The payment module gets the subscription code from the payment gateway and updates
 * the subscription in the KonaKart database to add the code.
 */
PaymentOptions options = new PaymentOptions();
options.setSubscriptionId(subscription.getId());
options.setOrderId(subscription.getOrderId());
options.setCreditCard((CreditCard) (subscription.getCreditCard()));
options.setAction(KKConstants.ACTION_CREATE_SUBSCRIPTION);
adEngineMgr.callPaymentModule(getEng().getEngConf(),
    "com.konakartadmin.modules.payment.authorizenet.AdminPayment", options);
```

This code creates a PaymentOptions object, fills it with the relevant ids and credit card information before passing it to the payment module in order to create the subscription.

Update a subscription

A subscription may need to be updated for various reasons. Common updates are to modify the amount that is billed at regular intervals and to add new credit card details when the current card expires.

In order to achieve this, the storefront application must gather the new credit card information from the customer and call the *updateSubscription()* API call in the same way as the *insertSubscription()* API call

was called above. The example call to AuthorizeNet is again in the OrderIntegrationMgr in the method called *afterUpdateSubscription()* . Here you can see that AuthorizeNet is called to update the subscription.

```
/*
 * Create a PaymentOptions object to pass to the method that calls the payment gateway.
 * The payment module gets the subscription code from the subscription object in the
 * KonaKart database which it looks up using the id in the PaymentOptions.
 */
PaymentOptions options = new PaymentOptions();
options.setSubscriptionId(subscription.getId());
options.setOrderId(subscription.getOrderId());
options.setCreditCard((CreditCard)(subscription.getCreditCard()));
options.setAction(KKConstants.ACTION_UPDATE_SUBSCRIPTION);
adEngineMgr.callPaymentModule(getEng().getEngConf(),
    "com.konakartadmin.modules.payment.authorizenet.AdminPayment", options);
```

This code creates a PaymentOptions object, fills it with the relevant ids and credit card information before passing it to the payment module in order to update the subscription.

Read the status of a subscription

AuthorizeNet allows you to detect the status of a subscription through the API. Using this functionality it's possible to create a batch program that checks the status for all active subscriptions in the KonaKart database in order to detect whether any require attention. If the credit card has expired, an eMail may automatically be sent by the batch program to the customer asking him to log into the store front application and provide new credit card information. Alternatively the payment gateway may generate reports or lists of accounts that require attention.

Manual Subscription Management

The Admin App allows you to manually insert and update subscriptions. Using the Custom Engine you can customize the following API calls to mirror the changes in the payment gateway.

- *InsertSubscription()* : The subscription object is inserted in the KonaKart database before calling the payment gateway. After having created the subscription, the payment gateway implementation should update the KonaKart subscription object in the database with the identifier generated by the payment gateway. This code is used in future communications with the payment gateway in order to identify the subscription.
- *UpdateSubscription()* : The KonaKart subscription object being updated contains the payment gateway subscription code that is used to identify the subscription saved by the payment gateway. This API call may be used for example to disable the subscription or to modify the subscription amount.
- *DeleteSubscription()* : The KonaKart subscription object being deleted contains the payment gateway subscription code that is used to identify the subscription saved by the payment gateway.

Managing Recurring Billing through KonaKart

In this case the credit card details are stored by KonaKart in an encrypted format within the Order object. The actual payments are made by a KonaKart Batch program that interfaces to the payment gateway to perform the payment transaction. An example of this batch is provided in source code format so that it can be modified. The batch is called recurringBillingBatch() and can be found within AdminOrderBatchMgr.java. It loops through all subscriptions that are active and that have a nextBillingDate equal to or later than the current date. Based on the data found within the Subscription object and attached PaymentSchedule

object, the batch program can make the payment and update the Subscription with a new nextBillingDate. The subscription object is also updated if the transaction with the payment gateway fails to indicate that there has been a problem.

The batch uses the *callPaymentModule* API call to communicate with the payment gateway. The source code example calls the admin payment module for AuthorizeNet (com.konakartadmin.modules.payment.authorizenet.AdminPayment).

Chapter 15. Multi-Vendor Functionality

KonaKart allows vendors to manage their own products and orders through the Admin App. The storefront application displays products from all vendors and allows the customer to checkout with any selection of products in a single order.

During the checkout process the products are grouped by vendor and the customer may select the appropriate shipping method for each vendor store from a list of those made available. Once the order has been confirmed by the customer, it is automatically split up so that each vendor receives an order for his own products. As the products are shipped by the vendor, the status of the vendor order (and maybe tracking number) is propagated to the parent order so that the customer only ever sees a single order with a history trail of events that occur during the lifecycle of the order.

Configuration

Multi-vendor mode is only valid when running KonaKart in multi-store shared products mode. It may be enabled through a configuration variable under Configuration >> Store Configuration.

Each vendor is allocated a KonaKart virtual store where products may be inserted and edited and each store may use different shipping modules which are selected from the modules available within the KonaKart system.

The storefront application uses the parent store or main store which is managed by an administrator. Typically the main store doesn't actually have any products of its own but it shares the products of the vendor stores.

Setting up a Main Store

A main store is managed by the administrator and has the following properties:

- It has no products. All products are shared with the vendor stores.
- It must have one or more payment modules defined since the payment process takes place on the main store.
- It has no shipping modules. All shipping is performed by the vendor stores.
- It shouldn't have a Tax Order Total module since this is created by the system as a sum of the individual vendor store tax amounts.
- It should have a Shipping Order Total module which will contain the sum of the individual shipping amounts from the vendor orders.

Order Total Modules	
Module Name	Sort Order
Buy X Get Y Free	•
Gift Certificate	•
Product Discount	•
Reward Points	•
Redeem Points	•
Shipping Discount	•
Shipping	30
Sub-Total	1
Tax	• ←
TaxCloud Tax	• ←
Total	40
Order Total Discount	•
Free Product	•

Main Store Order Total Modules (No Tax modules installed)

Setting up a Vendor Store

A vendor store is managed by the vendor and has the following properties:

- It contains the products sold by the vendor. These products are shared with the main store.
- It has no payment modules. Payment is through the main store.
- It must have one or more shipping modules defined.
- It has the standard Order Total modules including Tax and Shipping.

Edit Product - 77 - Vendor1 Product

[Description](#) [Details](#) [Images](#) [Attributes](#) [Quantities](#) [Categories](#) [Merchandising](#) [Prices](#) [Downloads](#) [Custom Tags](#) **Stores**

Vendor2 Store

Main Store

PId:	-1	C1:	<input type="text"/>	Delete
C2:	<input type="text"/>	C3:	<input type="text"/>	

>>

Vendor1 Store

PId:	-1	C1:	<input type="text"/>	Delete
C2:	<input type="text"/>	C3:	<input type="text"/>	

[Save](#) [Back](#)

Share vendor products with the Main Store

Order Total Modules

Module Name	Sort Order
Buy X Get Y Free	•
Gift Certificate	•
Product Discount	•
Reward Points	•
Redeem Points	•
Shipping Discount	•
Shipping	2
Sub-Total	1
Tax	•
TaxCloud Tax	39
Total	40
Order Total Discount	•
Free Product	•
Install	

Typical Vendor Store Order Total Modules

Orders

The customer, vendor and administrator all have different views of the order.

Customer View of the Order

Shopping Cart	
Vendor1 Store	
Item	Total
Vendor1 Product	\$25.00
Quantity: 1	
Tax:	\$2.70
Flat Rate (Shipping):	\$20.00
Flat Rate	<input type="button" value="▼"/>
Vendor2 Store	
Item	Total
Vendor2 Product	\$56.00
Quantity: 1	
PA Tax 10%:	\$5.60
Flat Rate (Shipping):	\$30.00
Sub-Total:	\$81.00
Shipping:	\$50.00
Tax:	\$8.30
TOTAL:	\$139.30

CONFIRM ORDER

Customer View of the Order

In the checkout screen example above, a customer is buying a product from Vendor1 Store and one from Vendor2 Store. Store 1 has multiple shipping options so the customer can choose the desired option from a list. Store 2 has a single shipping rate which is displayed. Store 1 has been configured to use the Tax Cloud tax module whereas Store 2 is using internal KonaCart tax calculation. The Sub-Total, Shipping, Tax and Total are displayed using the order total modules of the main order.

Vendor View of the Order

The Vendor can view the order from the Admin App after having logged into his store.

 **KonaKart**
Enterprise Java eCommerce

KonaKart - Order Details

Order Number:	185			
Date Ordered:	9-Jul-2013 12:28:10			
Customer: John doe 250 University Ave California, Pennsylvania 15419 United States	Shipping Address: John doe 250 University Ave California, Pennsylvania 15419 United States	Billing Address: John doe 250 University Ave California, Pennsylvania 15419 United States		
Telephone Number:	12345			
E-Mail Address:	doe@konakart.com			
Payment Method:	Cash on Delivery			
Product	Model	Qty.	Price	Total
Vendor2 Product	V2	1	\$56.00	\$56.00
		Sub-Total:	\$56.00	
		Flat Rate (Shipping):	\$30.00	
		PA Tax 10%:	\$5.60	
		Total:	\$91.60	

Vendor View of the Order

The vendor can only see the products that he is responsible for delivering.

Administrator view of the Order

The Administrator can see a full view of the order, including the details for each vendor store. This information allows the administrator to determine how to distribute the payment funds to the individual stores.

KonaKart
Enterprise Java eCommerce

KonaKart - Order Details

Order Number:	183	
Date Ordered:	9-Jul-2013 12:28:10	
<hr/>		
Customer: John doe 250 University Ave California, Pennsylvania 15419 United States	Shipping Address: John doe 250 University Ave California, Pennsylvania 15419 United States	Billing Address: John doe 250 University Ave California, Pennsylvania 15419 United States
<hr/>		
Telephone Number: 12345		
E-Mail Address: doe@konakart.com		
Payment Method: Cash on Delivery		

Vendor1 Store - store2				
Product	Model	Qty.	Price	Total
Vendor1 Product	v1	1	\$25.00	\$25.00
 Sub-Total: \$25.00				
Flat Rate (Shipping): \$20.00				
Tax: \$2.70				
Total: \$47.70				

Vendor2 Store - Store3				
Product	Model	Qty.	Price	Total
Vendor2 Product	v2	1	\$56.00	\$56.00
 Sub-Total: \$56.00				
Flat Rate (Shipping): \$30.00				
PA Tax 10%: \$5.60				
Total: \$91.60				
 Sub-Total: \$81.00				
Shipping: \$50.00				
Tax: \$8.30				
Total: \$139.30				

Administrator View of the Order

Order State Change

A method called manageMultiVendor() has been added to the AdminOrderIntegrationMgr under KonaKart\custom\adminapppn\src\com\konakartadmin\bl . The method serves as an example to show how state changes of vendor orders may be propagated to the main order.

This example implementation defines a couple of states (Delivered and Partially Delivered) and adds them to the status trail of the main order. If the new state of the vendor order is Delivered, then it looks at the current states of all of the vendor orders to see whether they have all been fully delivered. If this is the case then the state of the parent order is set to Delivered otherwise to Partially Delivered.

Order Payment

The customer pays for the order using the payment gateway of the main store so the full amount is received by the main store. It is the responsibility of the main store administrator to distribute the funds to the vendors, maybe keeping a percentage depending on the agreement between the vendor and main store.

Chapter 16. Custom Validation

Configuring the Validation used in KonaKart.

Custom Validation for the Store-Front

This section defines how you configure the custom validation for the KonaKart application

Configuring validation on data entered through the UI

The forms in the storefront application use jQuery validation to provide JavaScript based validation which doesn't require a screen refresh.

The internationalized validation error messages are retrieved from the message catalog and are defined in *MainLayout.jsp* . The message keys are all prefixed with *jquery.validator* .

The validation criteria for the form fields are found within the JavaScript file called *kk.validation.js* . After installation, this file may be found in the *KonaKart\webapps\konakart\script* directory.

Custom Validation for the Admin Application

This section defines how you configure the custom validation for the KonaKart Admin application

CustomValidation.properties file

From release 2.2.3.0 of KonaKart it is possible to modify the default field validation in certain parts of the Admin App.

A new properties file is provided, called *CustomValidation.properties*. You will find this in the classes directory of the konakartadmin webapp.

The format and description of this file is defined within the file itself, but an extract is as follows:

```

# -----
#
# K O N A K A R T   C U S T O M   V A L I D A T I O N
#
# -----
# Parameters to define custom validation in the KonaKart Admin App
# -----


# -----
# If no custom validations are defined (or this file is removed) the
# default validation rules are used in the KonaKart Admin App.
#
# If you define custom validations in this file they will override
# the default rules defined in the KonaKart Admin App.
# Therefore, you only need to define the custom validation rules
# that are different to the defaults.
# -----


# -----
# If your intention is to increase the number of characters allowed
# in the database for a certain quantity, you will have to modify
# the characteristics of that column in the database first. If you
# then wish to validate the attribute in the KonaKart Admin App to
# allow for the increased column width... you also need to add your
# custom validation to this file
# -----


# -----
# Format:
#
# ClassName.Attribute = [min];[max]
#
# If min or max are not specified they will not be checked.
#

```

Therefore, if you wanted to change the validation on the Product SKU field in the Admin App, you would specify:

Product.sku	= 1;18
-------------	--------

Fields Supported by Custom Validation

Only a certain subset of fields can have their validation overridden in this way, but if you need other fields to be made available for validation please get in touch with support at KonaKart.

All of these fields currently supported are listed in the CustomValidation.properties file. By default they are commented out.

CustomValidation Using a Custom Engine

If you have validation requirements that cannot be satisfied using the simpler properties-file-based techniques described above you can add custom validation using a Custom KKAdmin engine.

In order to do this you need to know which APIs are called when you save a record in the Admin App. Which API is used is usually obvious from the name (eg. editCustomer, updateCustomer etc) but if not you can establish this by enabling the DEBUG logging on the konakartadmin webapp. Once you know the interface move the Custom Engine implementation java source file from the "custom/adminengine/gensrc" directory tree to the "custom/adminengine/src" tree and rebuild your custom engine. Refer to the "Engine Customization" section in the User Guide for details).

For example, if you want to add some validation before saving an Order you should move ".\custom\adminengine\gensrc\com\konakartadmin\app\UpdateOrder.java" to ".\custom\adminengine\src\com\konakartadmin\app\UpdateOrder.java". (Other Order-saving APIs may also need to be moved and customised depending on what's being saved).

Having moved the source file you need to add your validation code. An example is:

```
public void updateOrder(String sessionId, int orderId, int orderStatus, String comments,
                        boolean notifyCustomer, AdminOrderUpdate updateOrder)
                        throws KKAdminException
{
    /**
     * Add your custom validation before the record is saved by updateOrder.
     */

    if (Your_Own_Custom_Validation(orderId) == false)
    {
        // Validation failed so we throw an exception back to the Admin App
        throw new KKAdminException("Validation Exception", true /* hideMoreDetails button */);
    }

    /**
     * Process normally because the validation was successful.
     */

    kkAdminEng.updateOrder(sessionId, orderId, orderStatus, comments, notifyCustomer,
                           updateOrder);
}
```

Note that when throwing an exception in your custom engine code you have the choice to "HideMoreDetails" (see the throw statement in the example above). This allows you to hide the default "More Details" button on the error message dialog box in the Admin App when this exception is caught. This allows you to hide technical validation details from the user (that are displayed by default when the "More Details" button is clicked) if you so wish.

Chapter 17. Custom Product Attributes and Miscellaneous Items

This chapter describes how to use Custom Attributes, Miscellaneous Items and Product Description Custom Fields

Using Custom Product Attributes

This section shows you how to create manage and display custom product attributes.

What types of Custom Attributes can be added to a product?

Every product in KonaKart can be assigned two types of custom attributes and an array of miscellaneous items. The custom attributes may be maintained in the Custom tab folder of the Edit Product Panel. The attributes in the Custom Attributes sub folder are mapped to database table attributes in the product table. This means that they can be added as constraints to product search queries and used to order the result of a search. The custom attributes consist of 10 strings, 2 integers and 2 decimal fields. Note that in the Admin App the labels for the entry fields may be customized using the message catalog.

The attributes in the Template Attributes sub folder are defined using the Custom Attributes and Custom Attribute Templates panels. These attributes have associated metadata which is used to:

- Decide what widget to use when entering data using the Admin App
- Store validation rules used by the Admin App to validate entered data.
- Store a message catalog key for each attribute.
- Store an attribute type and template used by the storefront application to dynamically detect the type of data being displayed and how to display it using the template.
- Custom fields which may be used by the storefront application to display the data. i.e. to group the custom attributes.

When saved, these attributes are encoded into an XML structure and saved within a single database attribute of the product table. This means that they may not be used as constraints for searching for products or ordering the array of products returned by a search.

Whenever a product is returned by an API call of the Admin or Application engine, these attributes are attached to the product within an array. Each element within the array contains the name, type and value of the attribute. It may also contain the template, message catalog key and custom fields if these have been defined.

The miscellaneous items are objects that have a type and a type description as well as a value. They can be used to add an array of items to any product or category. For example you may want to associate a product with a number of documents. In this case you define a "document" miscellaneous item type and a miscellaneous item (of type document) for each document.

Creating a Custom Attribute Definition and adding it to a Template

Custom attributes definitions may be managed using the Custom Attributes panel of the Admin App. The panel allows you to create new and manage existing attributes. Each custom attribute definition has the following attributes:

- *Name*: The unique name of the custom attribute which is a compulsory field. i.e. ScreenSize or MegaPixels etc.
- *Type*: You may choose types from a drop list. This information may be used by the storefront application when displaying the attribute value. i.e. If it knows the type of attribute it may decide what template to use in order to display it.
- *Set Function*: This is used by the Admin App when creating a UI widget to insert the custom attribute value once it has been added to a product. Valid Set Functions are:
 - integer(min,max) where min and max may be numbers or set to null. i.e. integer(10,null) checks that the integer has a minimum value of 10 or above. integer(null,null) just checks that the value entered is an integer without doing any range checking.
 - double(min,max) using the same logic as explained above for decimal numbers.
 - string(min,max) using the same logic as explained above for the length of the string.
 - choice('true','false')creates a radio button panel with the options of true and false. choice('label.small','label.medium','label.large') creates 3 radio buttons where the labels are looked up in a message catalog. In this case, the text displayed is looked up from the message catalog, whereas the values saved are always label.small, labl.medium and label.large regardless of the language.
 - option(0=date.day.long.Sunday,1=date.day.long.Monday,2=date.day.long.Tuesday) creates a drop list for the first 3 days of the week. The text of the days displayed is retrieved from the message catalog whereas the saved data is 0 for Sunday, 1 for Monday and 2 for Tuesday.
 - RichText(x) where x defines the vertical size (in elements) of the Rich Text data entry widget. (e.g. RichText(10) for a size of 10em)
- *Template*: The template contains a string that may be used by the storefront application to display the value. i.e. In the case of a date, the storefront application could detect that it is a date from the type and then use the template to display the date correctly.
- *Msg Cat Key*: This is used by the storefront and Admin applications to look up the label from the message catalog. If not present, the name may be used.
- *Validation*: This may contain regular expression such as true|false or [0-9]* which will be used by the admin app to validate the attribute.
- *Custom Fields*: The custom fields may contain any metadata for the attribute which can be used by the storefront application. i.e. A custom field may contain data useful for grouping custom attributes.

The default demo database comes complete with a number of example custom attribute definitions.

Custom attribute definitions may be used by more than one template. In order to assign an attribute to a product, it first needs to be added to a template since a product is assigned a template and not an array of custom attributes. Templates are managed using the Custom Attribute Templates panel. Once a template

has been created, it may be assigned a number of custom attributes using a pop-up select panel. The order in which the attributes are assigned to the template is important since this will be the order of the custom attributes within the attribute array attached to a product.

Entering Template based Custom Attribute data for a Product

In the Details folder of the Edit Product Panel, through the use of a pop-up panel, one or more templates may be chosen for a product. Note that the changes only take effect once the pop-up panel has been closed and the save button has been clicked. If any of the chosen templates has a number of custom attribute definitions then an equivalent number of custom fields will appear in the Template Attributes sub folder of the Custom folder. These entry fields which may include drop lists and radio buttons, allow you to enter the custom values for each product. The entry fields for the first template in the list are displayed first, followed by those of the second template etc.

When the product is saved, the custom field data is encoded within an XML structure and saved within an attribute of the product table in the database.

Displaying the custom data in the storefront Application

If a product includes custom template data, when fetched from the KonaKart Application engine, it will contain an instantiated *customAttrArray* with the custom attribute values and metadata. Note that the product will also contain an attribute called *customAttrs* that represents the XML of the custom data. If preferred, you may display the custom data directly from the XML.

Miscellaneous Items

The Admin App contains a panel for maintaining miscellaneous item types, and one for miscellaneous items. Miscellaneous items will typically be documents or videos that you want to associate with a product or category. Each product or category can have an unlimited number of miscellaneous items which are returned in an array connected to the object. When using an API call that returns an array of products, you can define in the DataDescriptor object whether you want the miscellaneous items to be populated or not. When returning a single product or category, the array is populated by default.

The miscellaneous item type panel allows you to define item types where the name and description of the type needs to be entered for all supported languages. Once item types have been defined, you can then insert the miscellaneous items for a product or category by clicking on the "Misc Items" button in the Products or Categories panel after having selected a product or category.

Product Description Custom Fields

From v6.6.0.0 of KonaKart it is possible to use custom fields as part of Product Descriptions. The advantage of these custom fields is that they can be defined for each language you support in the system (as with the other attributes on Product Descriptions such as Name, Description, URL etc).

By default these custom fields are created to use very little space in the database. The reason for this is to optimise overall KonaKart performance for those users who do not wish to use these fields. For those users who do wish to use these fields and need them to be able to hold more data in each field it is possible to increase the size of these columns in the database as required. All that is required is to execute an SQL statement appropriate for the database you are using. Therefore, to enlarge customd4 to 2000 bytes for a MySQL database you should execute:

```
ALTER TABLE products_description MODIFY customd4 VARCHAR(2000);
```

Having extended the sizes of the these columns the KonaKart APIs will continue to work in the same way - except that they will return the extended data items from these columns.

By default the product description custom fields are not displayed in the Admin App. To maintain these product description custom fields in the Admin App you need to enable the "showing" of these using File-Based Configuration (Enterprise-Only - in the konakartadmin_gwt.properties file). You can enable all of the custom fields or just the fields that you require:

```
# Product Description Custom fields are all hidden by default
# The Admin App displays fields 1-3 in 1-line textboxes and fields 4-6 in multi-line edit areas.
#fbc.kk_panel_editProduct.desc.show_custom1          = true
#fbc.kk_panel_editProduct.desc.show_custom2          = true
#fbc.kk_panel_editProduct.desc.show_custom3          = true
#fbc.kk_panel_editProduct.desc.show_custom4          = true
#fbc.kk_panel_editProduct.desc.show_custom5          = true
#fbc.kk_panel_editProduct.desc.show_custom6          = true
```

By default the product description custom fields are validated according to their default sizes (and allowing empty fields). To change the validation for these fields to suit your needs, set the required validation in the CustomValidation.properties file for example (set these as required and uncomment the respective lines):

```
# Product.desc.custom1      = 0;128
# Product.desc.custom2      = 0;128
# Product.desc.custom3      = 0;164
# Product.desc.custom4      = 0;2000
# Product.desc.custom5      = 0;2000
# Product.desc.custom6      = 0;2000
```

Chapter 18. Programming Guide

A guide to programming and customizing KonaKart.

One of the most powerful aspects of KonaKart is the flexibility in the number of different ways it can be customized. This chapter contains descriptions of some of these techniques.

Using the Java APIs

One of the most important features of KonaKart is the availability of a set of open APIs that allow you to control KonaKart as you require as part of your IT solution. You may wish to add a different front-end for your KonaKart shopping cart application or you may wish to integrate KonaKart with other applications in your back office.

When you use the java APIs for whatever purpose you can be confident that they will work with future releases of KonaKart as backwards compatibility will always be maintained (although some API calls may be deprecated, they will not be removed).

A number of simple java sources are provided in every KonaKart download kit (you can find them under the installation directory in a directory called `java_api_examples`) that demonstrate how you call the direct POJO form of the KonaKart APIs:

To give you a flavor of the examples in the downloadable sources, here is an extract from one of them that demonstrates how you can register a new customer using the KonaKart Admin APIs. (See the example sources for a complete set of java files).

```

// Instantiate an AdminCustomerRegistration object and set its
// attributes

AdminCustomerRegistration acr = new AdminCustomerRegistration();

// Compulsory attributes

// Gender should be set to "m" or "f" (or "x" if "Other Genders" are enabled)

acr.setGender("m");
acr.setFirstName("Peter");
acr.setLastName("Smith");

// If you do not require the birthdate, just set it to the current
// date. It is compulsory and so needs to be set.

acr.setBirthDate(new Date());
acr.setEmailAddr("peter.smith@konakart.com");
acr.setStreetAddress("23 Halden Close");
acr.setCity("Newcastle");
acr.setPostcode("ST5 0RT");
acr.setTelephoneNumber("01782 639706");
acr.setPassword("secret");

// Optional attributes

acr.setCompany("DS Data Systems Ltd.");

// Country Id needs to be set with the id of a valid country from
// the Countries table

acr.setCountryId(222);
acr.setFaxNumber("01782 639707");

// Newsletter should be set to "0" (don't receive) or "1" (receive)
acr.setNewsletter("1");

acr.setSuburb("May Bank");
acr.setState("Staffs");

// Optional Custom fields for customer object

acr.setCustomerCustom1("Number Plate");
acr.setCustomerCustom2("Driver's license");
acr.setCustomerCustom3("Passport");
acr.setCustomerCustom4("custom 4");
acr.setCustomerCustom5("custom 5");

// Optional Custom fields for address object

acr.setAddressCustom1("custom 1");
acr.setAddressCustom2("custom 2");
acr.setAddressCustom3("custom 3");
acr.setAddressCustom4("custom 4");
acr.setAddressCustom5("custom 5");

// Register the customer and get the customer Id

int custId = eng.registerCustomer(sessionId, acr);

System.out.println("Id of the new customer = " + custId);

```

Using the SOAP Web Service APIs

KonaCart has four major interface types - the direct POJO interface, the SOAP interface, and, if you have the Enterprise Extensions version, the JSON interface and the RMI interface. (JSON support is only available for the application engine and not the Admin engine). Each of these support exactly the same interface calls. Which one you choose will depend on many factors but the best performance is likely to come from the direct interface, whereas the SOAP, RMI and JSON interfaces give greater flexibility

in distributed implementations and inter-operability. Using JSON over low-bandwidth networks and on mobile devices will probably give the best performance in these environments.

It's remarkably simple to use KonaKart's SOAP interfaces. Although it is easier to use the SOAP engines that are already available in the KonaKart jars (see later for the recommended approach) if your IDE can create client stubs from WSDL then you can use that to create your SOAP clients or, if you prefer, you can use AXIS as in the downloadable example which follows.

There are two WSDL definitions; one for the KonaKart Application API, and the other for the KonaKart Admin API. They can be seen at these URLs:

- <http://www.konakart.com/konakart/services/KKWebServiceEng?wsdl> [http://www.konakart.com/konakart/services/KKWebServiceEng?wsdl]
- <http://www.konakart.com/konakartadmin/services/KKWSAdmin?wsdl> [http://www.konakart.com/konakartadmin/services/KKWSAdmin?wsdl]

Enable the SOAP Web Services

Since version 3.2.0.0 of KonaKart, the SOAP APIs are disabled during the installation process. The purpose of this decision is to make the KonaKart engines more secure. The process for enabling the APIs is very simple and can be achieved by running an ant target called *enableWebServices* in the *KonaKart/custom* directory as shown below. (An equivalent ANT task called *enable_JSON* is provided to enable JSON).

```
C:\Program Files\KonaKart\custom>.\bin\ant enableWebServices
Buildfile: build.xml

enableWebServices:
[echo] Allow all methods in WSDD files:

BUILD SUCCESSFUL
Total time: 0 seconds
```

The ant target modifies the files called *server-config.wsdd* present in the WEB-INF directory of the KonaKart store front and admin applications. The modification is simple:

```
The line
<parameter name="allowedMethods" value="none" />
is changed to
<parameter name="allowedMethods" value="*" />
```

Securing the SOAP Web Services

KonaKart web services use the AXIS 1 web services framework in a standard manner. This allows you to add handlers in the request and response pipelines of the web service message processors for a variety of purposes. Two common examples are for web service monitoring and security.

A step-by-step guide to implementing WS-Security for the KonaKart web services is provided in the download kits under *java_soap_examples* in a document called *KonaKart-WS-Security.txt*.

Step-by-step guide to using the SOAP APIs:

1. The SOAP examples kit that is provided in every KonaKart download kit (you can find them under the installation directory in a directory called `java_soap_examples`) contains everything you need to build a simple SOAP client from the KonaKart Application WSDL. First verify that you have the kit in your version of KonaKart.
2. Next, ensure that KonaKart is running on your machine. After the default installation you should be able to start KonaKart by clicking on the "Start KonaKart Server" start-up icon (on Windows) or by executing the `{Konakart Home}/bin/startkonakart.sh` script (on *nix). You need KonaKart to be running in order to get responses so your SOAP calls that you will make with the examples.
3. Check that the KonaKart server has started OK - a quick way would be to see if the application appears at `http://localhost:8780/konakart` [`http://localhost:8780/konakart`]
4. Open up a console window and change directory to the location of the `java_soap_examples`, for example this would be:

C:\> cd C:\Program Files\KonaKart\java_soap_examples on Windows.

5. Ensure that the environment variable `JAVA_HOME` is set correctly. On Windows, a suitable setting might be:

JAVA_HOME=C:\jdk1.6.0_22

6. Ensure that the `JAVA_HOME/bin` directory appears on your PATH. On Windows, this might look something like this:

Path=C:\jdk1.6.0_22\bin;C:\WINDOWS\system32;C:\WINDOWS

{You could also check that when you issue "java -version" you get the java version you expect}

7. If you already have ANT installed, `ANT_HOME` is defined and ANT's bin directory is on your PATH you can skip the next two sections. If you don't have ANT installed you can use the cut-down version of ANT that we include in the KonaKart download kit, but you must set the following environment variables. Ensure that the environment variable `ANT_HOME` is set correctly in the command shell that you are executing (you probably don't want to change the environment variable globally). On Windows, if you installed KonaKart to the default location (`C:\Program Files\KonaKart`), it would be (note the slashes):

ANT_HOME=C:/Program Files/KonaKart/custom/bin

8. Ensure that the `ANT_HOME/bin` directory appears on your PATH. On Windows, this might look something like this:

Path=C:\jdk1.6.0_22\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\KonaKart\custom\bin

As a sanity check that you are set up ready to build the examples, check that you see the following when you issue a "ant -p" command:

```
C:\Program Files\KonaKart\java_soap_examples>ant -p
Buildfile: build.xml

Main targets:

axis_client_generation  Generate client stubs from the WSDL
build                  Creates the SOAP clients, compiles and runs a little example
clean                  Clears away everything that's created during a build
compile                Compile the examples
run                   Run the little example program

Default target: build
```

[The KonaKart download kit contains just enough of ANT for you to build the examples. For subsequent development using ANT you should consider installing a complete ANT installation - check the Apache ANT site [<http://ant.apache.org/>] for details]

- Simply execute the "ant" command to compile, then run, the simple SOAP example.

If all is well you should see the following output:

```
C:\Program Files\KonaKart\java_soap_examples>..\custom\bin\ant
Buildfile: build.xml

clean:
[echo] Cleanup...

axis_client_generation:
[echo] Create the KonaKart client stubs from the WSDL

compile:
[echo] Compile the examples
[mkdir] Created dir: C:\Program Files\KonaKart\java_soap_examples\classes
[javac] Compiling 59 source files to C:\Program Files\KonaKart\java_soap_examples\classes
[javac] Note: Some input files use unchecked or unsafe operations.
[javac] Note: Recompile with -Xlint:unchecked for details.

run:
[java] First the low-level version...
[java]
[java] There are 239 countries
[java] There are 10 manufacturers
[java] The default currency is: USD
[java] 29 products found
[java] 29 length of product array
[java]
[java] Next the recommended high-level version...
[java]
[java] There are 239 countries
[java] There are 10 manufacturers
[java] The default currency is: USD
[java] 29 products found
[java] 29 length of product array

build:
BUILD SUCCESSFUL
Total time: 10 seconds
```

The code is very simple; here are a couple of extracts from AxisExample.java. This is the lower-level version:

```

KKWSEngIf port = new KKWSEngIfServiceLocator().getKKWebServiceEng();

// make some example calls

Country[] countries = port.getAllCountries();
System.out.println("There are " + countries.length + " countries");

Manufacturer[] manufacturers = port.getAllManufacturers();
System.out.println("There are " + manufacturers.length + " manufacturers");

Currency curr = port.getDefaultCurrency();
System.out.println("The default currency is: " + curr.getCode());

```

This is the recommended higher-level version:

```

EngineConfigIf engConfig = new EngineConfig();
engConfig.setMode(EngineConfig.MODE_SINGLE_STORE);

KKEngIf engine = new KKWSEng(engConfig);

// make some example calls

CountryIf[] countries = engine.getAllCountries();
System.out.println("There are " + countries.length + " countries");

ManufacturerIf[] manufacturers = engine.getAllManufacturers();
System.out.println("There are " + manufacturers.length + " manufacturers");

CurrencyIf curr = engine.getDefaultCurrency();
System.out.println("The default currency is: " + curr.getCode());

```

Notes:

- If you've installed to a port other than 8780 and you wish to generate your client code you will have to modify the *konakart.wsdl* file to match (change the port number right at the end of the file) then re-run the client stub generation in the ANT file as above.
- If you've installed to a port other than 8780 but do not wish to generate your client code (as in the case of using the recommended higher-level technique) you will have to modify the *konakart_axis_client.properties* and *konakartadmin_axis_client.properties* file to define your SOAP endpoint locations.
- The files *konakart_axis_client.properties* and *konakartadmin_axis_client.properties* are used to define the locations of the respective web services. Therefore if these locations are not the default URLs you will need to modify these files as appropriate.
- If you get "connection refused" - check that your firewall isn't blocking your client calls and that the endpoints defined in the *konakart_axis_client.properties* and *konakartadmin_axis_client.properties* are actually responding correctly.
- Remember you have to enable the web services - see above for enabling the web services .

Using the RMI APIs

With the Enterprise Extensions version of KonaCart you have the ability to communicate with its engines using RMI. The KonaCart Application Engine is registered in an RMI Registry at a specified

port with the name "konakart.kkeng" and the KonaKart Admin Engine is registered with the name "konakart.kkadmineng".

By default the RMI Server servlet sections in the respective web.xml files for the konakart and konakartadmin webapps are commented out so no RMI services are enabled.

When enabled in the respective web.xml files, by default the RMI Registry is created on port 8790 and the two engines are bound to the above names.

Once enabled in the web.xml file, this default behavior can be controlled using the following configuration options in the respective webapp's web.xml: (the first one here is for the konakart webapp):

```
<!-- Servlet for RMI Server -->
<servlet>
    <servlet-name>KonakartRMIServlet</servlet-name>
    <display-name>KonaKart RMI Server</display-name>
    <description>KonaKart RMI Server</description>
    <servlet-class>
        com.konakart.rmi.KKRMServer
    </servlet-class>
    <init-param>
        <!-- The port number where the RMI registry will listen -->
        <param-name>port</param-name>
        <param-value>8790</param-value>
    </init-param>
    <init-param>
        <!-- Enable or Disable the RMI interface -->
        <param-name>rmiEnabled</param-name>
        <param-value>true</param-value>
    </init-param>
    <load-on-startup>20</load-on-startup>
</servlet>
```

Very similar configuration options are available for the RMI version of the KonaKart Admin Engine. The konakartadmin webapp's web.xml contains this servlet definition:

```
<!-- Servlet for RMI Server -->
<servlet>
    <servlet-name>KonakartAdminRMIServlet</servlet-name>
    <display-name>KonaKartAdmin RMI Server</display-name>
    <description>KonaKartAdmin RMI Server</description>
    <servlet-class>
        com.konakartadmin.rmi.KKRMSIAAdminServer
    </servlet-class>
    <init-param>
        <!-- The port number where the RMI registry will listen -->
        <param-name>port</param-name>
        <param-value>8790</param-value>
    </init-param>
    <init-param>
        <!-- Enable or Disable the RMI interface -->
        <param-name>rmiEnabled</param-name>
        <param-value>true</param-value>
    </init-param>
    <load-on-startup>20</load-on-startup>
</servlet>
```

Both the application and admin engines are bound in the same RMI Registry (assuming the port numbers remain the same in the respective web.xml configuration files) which is created to listen at the specified port if it isn't found.

You need to specify the mode that the RMI engines will use in the konakart.properties and konakartadmin.properties files in their respective webapps.

The configuration settings must match those of your KonaKart installation (the engine mode, customers shared and products shared settings). These are set by the installer automatically but if you have installed manually you need to adjust these to suit your installation:

```

# -----
# KonaKart engine class used by the RMI services
# For the default engine use: com.konakart.app.KKEng
# For the custom engine use: com.konakart.app.KKCustomEng

konakart.app.rmi.engine.classname = com.konakart.app.KKEng

# -----
# Enterprise Feature
# Engine mode that the RMI engine will use
# 0 = Single Store (default)
# 1 = Multi-Store Multiple-Databases (add konakart.databases.used above as well)
# 2 = Multi-Store Single Database

konakart.rmi.mode = 0

# -----
# Enterprise Feature
# Customers Shared / Products Shared mode that the RMI engine will use
# When in multi-store single database mode, the customers can be shared between stores

konakart.rmi.customersShared = false

# When in multi-store single database mode, the products can be shared between stores

konakart.rmi.productsShared = false

# When in multi-store single database mode, the categories can be shared between stores

konakart.rmi.categoriesShared = false

# -----
# RMI Registry Location - This is used to locate (not create) the RMI Registry
# The definition for the port that the RMI Registry will listen on is in the web.xml

konakart.rmi.host = localhost
konakart.rmi.port = 8790

```

Note that the values at the bottom (port and host) are used to locate the RMI Registry and not to define where it will be created. (Hence you could consider this a "client-side" definition for using your RMI services). See the above section that explains that the port number where the RMI Registry is created is declared in the web.xml.

A very similar definition exists in the konakartadmin.properties file for the KonaKart Admin RMI Engine definiton:

```
# -----
# KonaKart engine class used by the Admin RMI services
# For the default engine use: com.konakartadmin.bl.KKAdmin
# For the custom engine use: com.konakartadmin.app.KKAdminCustomEng

konakart.admin.rmi.engine.classname = com.konakartadmin.bl.KKAdmin

# -----
# Enterprise Feature
# Engine mode that the RMI engine will use
# 0 = Single Store (default)
# 1 = Multi-Store Multiple-Databases (add konakart.databases.used above as well)
# 2 = Multi-Store Single Database

konakart.rmi.mode = 0

# -----
# Enterprise Feature
# Customers Shared / Products Shared mode that the RMI engine will use
# When in multi-store single database mode, the customers can be shared between stores

konakart.rmi.customersShared = false

# When in multi-store single database mode, the products can be shared between stores

konakart.rmi.productsShared = false

# When in multi-store single database mode, the categories can be shared between stores

konakart.rmi.categoriesShared = false

# -----
# RMI Registry Location - This is used to locate (not create) the RMI Registry
# The definition for the port that the RMI Registry will listen on is in the web.xml

konakart.rmi.host = localhost
konakart.rmi.port = 8790
```

As with the SOAP versions of the engines, it's easy to select which one you would like to use in your code by simply specifying it by name at runtime.

A very simple example of this is provided in the GetProduct.java example provided in the download kit (under the java_api_examples directory). The example demonstrates how simple it is to select different engines and get the same result with each:

```

/*
 * Get an instance of the KonaKart engine and retrieve. The method called can be found in
 * BaseApiExample.java
 */
EngineConfig engConf = new EngineConfig();
engConf.setMode(getEngineMode());
engConf.setstoreId(getstoreId());
engConf.setCustomersShared(isCustomersShared());
engConf.setProductsShared(isProductsShared());
engConf.setCategoriesShared(isCategoriesShared());

/*
 * Instantiate a direct java Engine by name - to find a product
 * KKEngIf eng = new KKEng(engConf);
 */
KKEngIf eng = getKKEngByName("com.konakart.app.KKEng", engConf);
System.out.println("Get a product using the KKEng engine");
getProductUsingEngine(eng);

/*
 * Instantiate a java RMI Engine by name - to find a product
 * KKAdminIf rmiEng = new KKAdminIf(engConf);
 */
KKEngIf rmiEng = getKKEngByName("com.konakart.rmi.KKRMIEng", engConf);
System.out.println("Get a product using the KKRMIEng engine");
getProductUsingEngine(rmiEng);

```

Notice how both of the engines implement the KKEngIf. If you code your solution to the KKEngIf interface (and KKAdminIf interface for the Admin Engine) you can delay the decision about which engine to use until runtime or (probably more likely) until you have decided how you wish to distribute your KonaKart solution over multiple machines.

The implementation of the getKKEngByName() call used in the above example is provided in the GetProduct.java source file.

Since all three of the application engines implement the same KKEngIf interface the javadoc is applicable for every one. The equivalent is true for the KKAdminIf javadoc.

Using the JSON APIs

The Enterprise version of KonaKart offers you the ability to communicate with the application engine using JSON (JavaScript Object Notation). A servlet receives requests in JSON format over HTTP/HTTPS and passes these on to an application engine that processes the request. The response is returned in JSON format.

By default the JSON Server servlet sections in the web.xml file for the konakart webapp are commented out so JSON services are disabled after a standard install.

A convenient way to enable JSON services is to run the enable_JSON ANT task provided in the build.xml file in the custom directory of the standard installation as follows:

```
C:\Program Files\KonaKart\custom>bin\kkant enable_JSON
Buildfile: build.xml

enable_JSON:
enable_JSON_warning:
enable_JSON_enterprise:
[echo] Fix konakart web.xml to start-up JSON

BUILD SUCCESSFUL
Total time: 0 seconds
```

There are two sections in the konakart webapp's web.xml file to modify to enable the JSON server. The first is the servlet definition and the second is the servlet mapping. To enable the JSON Server servlet section you must uncomment the definition in the konakart webapp's web.xml file shown below:

```
<!-- JSON Server -->
<servlet>
  <servlet-name>KonaKart_JSON_Servlet</servlet-name>
  <display-name>KonaKart JSON Server</display-name>
  <description>KonaKart JSON Server</description>
  <servlet-class>
    com.konakart.json.KKJSONServer
  </servlet-class>
  <init-param>
    <param-name>jsonEnabled</param-name>
    <param-value>true</param-value>
  </init-param>
  <init-param>
    <param-name>includedInterfaces</param-name>
    <param-value></param-value>
  </init-param>
  <init-param>
    <param-name>excludedInterfaces</param-name>
    <param-value></param-value>
  </init-param>
  <load-on-startup>30</load-on-startup>
</servlet>
```

Note the availability of the two JSON servlet parameters: "excludedInterfaces" and "includedInterfaces". These can be used to fine tune the JSON services that you make available from your system. It is recommended that you only make available those interfaces that are required by authorised client applications. In both cases (excludedInterfaces and includedInterfaces) you simply specify a comma separated list of KKEngIf interfaces that are to be allowed or disallowed. For a list of KKEngIf interfaces available in your version of KonaKart you will find a list in the comments in the server-config.wsdd file in your konakart/WEB-INF/ directory after a standard installation (provided from version 6.3.0.0). See the web.xml for the konakart webapp for more details.

Next, to enable the servlet mapping, you must uncomment the definition shown below. The servlet mapping may be set as you wish (the URL used must be used by the clients who wish to call the JSON Server and is defined in konakart.properties for clients using the client side of the JSON engine). The default servlet mapping URL is "/konakartjson" but this can be changed if required:

```
<!-- JSON Server -->
<servlet-mapping>
  <servlet-name>KonaKart_JSON_Servlet</servlet-name>
  <url-pattern>/konakartjson</url-pattern>
</servlet-mapping>
```

You need to specify the mode that the JSON engines will use in the konakart.properties file in the konakart webapp's WEB-INF/classes directory. This section defines the engine class used to service the JSON requests with the default being defined as com.konakart.app.KKEng.

The configuration settings must match those of your KonaKart installation (the engine mode, customers shared, products shared and categories shared settings). These are set by the installer automatically but if you have installed manually you need to adjust these to suit your installation:

```
# -----
# Enterprise Feature
# KonaKart engine class used by the JSON services
# For the default engine use: com.konakart.app.KKEng
# For the custom engine use: com.konakart.app.KKCustomEng

konakart.app.json.engine.classname = com.konakart.app.KKEng

# -----
# Enterprise Feature
# URL for the JSON engine servlet

konakart.json.engine.url = http://localhost:8780/konakart/konakartjson

# Generate match Id on generated JSON Requests

konakart.json.generateMatchIds = false

# -----
# Enterprise Feature
# Engine mode that the JSON engine will use
# 0 = Single Store (default)
# 1 = Multi-Store Multiple-Databases (add konakart.databases.used above as well)
# 2 = Multi-Store Single Database

konakart.json.mode = 2

# -----
# Enterprise Feature
# Customers Shared / Products Shared mode that the JSON engine will use
# When in multi-store single database mode, the customers can be shared between stores

konakart.json.customersShared = false

# When in multi-store single database mode, the products can be shared between stores

konakart.json.productsShared = false

# When in multi-store single database mode, the categories can be shared between stores

konakart.json.categoriesShared = false
```

Note that the property "konakart.json.engine.url" that defines the URL for the JSON service must match the web.xml servlet-mapping defintion (see above).

The property "konakart.json.generateMatchIds" defines whether matching Ids are generated in JSON requests that are sent to the engine. The default for this is false which minimizes message size and parsing time.

As with the POJO, RMI and SOAP versions of the engines, it's easy to select which one you would like to use in your code by simply specifying it by name at runtime.

A very simple example of this is provided in the GetProduct.java example provided in the download kit (under the java_api_examples directory). The example demonstrates how simple it is to select different engines and get the same result with each:

```

/*
 * Get an instance of the KonaKart engine and retrieve. The method called can be found in
 * BaseApiExample.java
 */
EngineConfig engConf = new EngineConfig();
engConf.setMode(getEngineMode());
engConf.setstoreId(getstoreId());
engConf.setCustomersShared(isCustomersShared());
engConf.setProductsShared(isProductsShared());
engConf.setCategoriesShared(isCategoriesShared());

/*
 * Instantiate a direct java Engine by name - to find a product
 * KKEngIf eng = new KKEng(engConf);
 */
KKEngIf eng = getKKEngByName("com.konakart.app.KKEng", engConf);
System.out.println("Get a product using the KKEng engine");
getProductUsingEngine(eng);

/*
 * Instantiate a java JSON Engine by name - to find a product
 * KKJSONEng jsonEng = new KKJSONEng(engConf);
 */
KKJSONEng jsonEng = getKKEngByName("com.konakart.json.KKJSONEng", engConf);
System.out.println("Get a product using the KKJSONEng engine");
getProductUsingEngine(jsonEng);

```

Notice how both of the engines implement the KKEngIf. If you code your solution to the KKEngIf interface you can delay the decision about which engine to use until runtime or (probably more likely) until you have decided how you wish to distribute your KonaKart solution over multiple machines.

The implementation of the getKKEngByName() call used in the above example is provided in the GetProduct.java source file.

Since all four of the application engines implement the same KKEngIf interface the javadoc is applicable to every one of them.

You do not have to use the client side of the JSON engine if you don't want to. Instead you may wish to construct your own client requests yourself. You can derive the format of the requests by applying the standard conventions for translating our KKEngIf into JSON calls as follows.

- "f" : Required. The function name, eg. "getProduct".
- "s" : Optional. The storeId (required in a multi-store environment) eg. "store1".
- "i" : Optional. The matchingId (for matching requests with responses - this id is returned in response messages if specified in the request) eg. "1234".
- input parameters : Required (they must match the names of the input parameters defined in KKEngIf). Standard JSON format for the input parameters if these are present.
- nulls : attributes with null values can be left out of the request. Attributes with null values are left out of all response messages.

Some examples of JSON requests to the KonaKart JSON Server:

```

A login request:

{
  "emailAddr" : "fred@flintstone.com",
  "password" : "abcde02",
  "f" : "login",
  "s" : "store1"
}

A customer registration request:

{
  "custReg" : {
    "emailAddr" : "robin@batcave.com",
    "city" : "GothamCity",
    "company" : "ACME Inc",
    "countryId" : 223,
    "faxNumber" : "123456",
    "firstName" : "Robin",
    "gender" : "m",
    "lastName" : "Wayne",
    "newsletter" : "0",
    "postcode" : "st5 ort",
    "productNotifications" : -1,
    "streetAddress" : "12345 Alligator Alley",
    "streetAddress1" : "Blue house",
    "suburb" : "Harlem",
    "telephoneNumber" : "654321",
    "birthDate" : 1303223768362,
    "addressCustom1" : "addressCustom1",
    "addressCustom5" : "addressCustom5",
    "customerCustom1" : "customerCustom1",
    "customerCustom5" : "customerCustom5",
    "groupId" : -1,
    "zoneId" : -1,
    "invisible" : false,
    "state" : "Florida",
    "password" : "abcde02"
  },
  "f" : "registerCustomer",
  "s" : "store1"
}

```

Some examples of JSON responses from the KonaCart JSON Server:

```

A response to a login request showing the result which is a sessionId:

{
  "r" : "f663832c049322d2fb6882ba0abf4db9"
}

A response from a getCustomer request:

{
  "r" : {
    "id" : 123,
    "type" : 0,
    "birthDate" : 1303223708000,
    "emailAddr" : "robin@batcave.com",
    "firstName" : "Robin",
    "gender" : "m",
    "lastName" : "Wayne",
    "telephoneNumber" : "654321",
    "invisible" : 0,
    "numberOfLogons" : 0,
    "defaultAddrId" : 147,
    "globalProdNotifier" : 0,
    "groupId" : -1
  }
}

```

If an exception is thrown during the processing of a request, the server will respond with a message in JSON format as follows:

```
An exception thrown in any request where a sessionId is required but cannot
be found is transformed into this JSON response:
```

```
{
    "e" : "com.konakart.app.KKException",
    "m" : "The session bad-session-id cannot be found"
}
```

```
An exception thrown in a login request is transformed into this JSON response:
```

```
{
    "e" : "com.konakart.app.KKException",
    "m" : "[5] Cannot find customer with email address = bad-user@localhost"
}
```

An optional JSON Administration server can be enabled to provide programmatic control of the JSON server. By default this is disabled in the konakart webapp's web.xml. This can be used to enable or disable the JSON server and also to set the included and excluded interfaces to be supported. Note that changes made via the JSON Administration server take effect in real time but are not persisted so the definitions specified in the konakart webapp's web.xml file will take effect on restart of the application.

```
<!-- Servlet for JSON Admin -->
<servlet>
<!--

Uncomment the section below if you want to use the JSON Admin Servlet

When sending these commands the password must match the one defined in the
"password" servlet parameter below.

Only enable the JSON Admin server if you need to and if you do, change the
password.

JSON Admin commands:
?cmd=enableJSON&pwd=password
    Enables the JSON server
?cmd=disableJSON&pwd=password
    Disables the JSON server
?cmd=excludeInterfaces&pwd=password&Interfaces=A,B,C
    Sets the excludedInterfaces to a comma separated list of KKEngIf interfaces
?cmd=includeInterfaces&pwd=password&Interfaces=A,B,C
    Sets the includedInterfaces to a comma separated list of KKEngIf interfaces
-->

<!-- JSON Admin
<servlet>
<servlet-name>KonaKart_JSON_Admin</servlet-name>
<display-name>KonaKart JSON Admin</display-name>
<description>KonaKart JSON Admin</description>
<servlet-class>
    com.konakart.json.KKJSONServerAdmin
</servlet-class>
<init-param>
    <param-name>password</param-name>
    <param-value>jason</param-value>
</init-param>
    <load-on-startup>29</load-on-startup>
</servlet>
End of JSON Admin -->
```

Running Your Own SQL

A convenient way to use the same database connection techniques as KonaCart is as follows:

First a *SELECT* query... note the processing of the result set after the results are returned.

```
/*
 * Run a query that selects the product id and product model from
 * all products that have an id less than 10. All Select queries
 * need to be run using the KKBasePeer.executeQuery command
 */

List<Record> records = KKBasePeer
    .executeQuery("select products_id, products_model, " +
        "products_price " +
        "from products " +
        "where products_id < 10");

/*
 * Loop through the result set and print out the results. Note that
 * the first attribute in the Record object is at index = 1 and not
 * index = 0
 */

if (records != null)
{
    for (Iterator<Record>
        iterator = records.iterator();
        iterator.hasNext();)
    {
        Record rec = (Record) iterator.next();
        System.out.println("id = " + rec.getValue(1).asInt()
            + "; model = "
            + rec.getValue(2).asString() + "; price = "
            + rec.getValue(3).asBigDecimal(/* scale */2));
    }
}
```

Here's a simple *UPDATE* example:

```
/*
 * Now let's run another query to change the model name of the
 * product with id = 1.
 *
 * All non select queries need to be run using the
 * KKBasePeer.executeUpdate command
 */

KKBasePeer.executeUpdate(
    "update products set products_model='Super Turbo' where products_id=1");
```

The */java_api_examples/src/com/konakart/apiexamples/RunCustomQuery.java* example, provided in the download kit, illustrates this technique.

Database Layer Changes from v7.2.0.0

In v7.2.0.0 KonaCart was upgraded to use Apache Torque 4 as its database layer replacing the previous Torque 3.3. This provides performance improvements and keeps KonaCart up-to-date with the latest stable version of Apache Torque.

KonaCart uses Torque (and parts of Village) in order to maintain portability across 5 databases and ensure that the highest possible performance is achieved with fine-grained control over the SQL that is executed.

(This is not always the case with alternative object-based persistence layers that tend to generate and execute redundant and less efficient SQL).

In addition to introducing a new version of Torque in v7.2.0.0, KonaKart has also undergone a number of changes, that you may need to be aware of, associated with the database and programming using the KonaKart/Torque persistence layer.

Table and Column Name Changes

In order to simplify portability across the supported databases, a few table and column names have been modified so that they are identical across all platforms. This change also allows a slight performance gain as KonaKart does not have to calculate the different platform-specific names at runtime as it did in previous versions.

The "old" table and column names stated below are the ones from MySQL. They will have been truncated for other databases such as Oracle and DB2. In future all table and column names will be the same across all databases.

Old Table Name	New Table Name
products_options_values_to_products_options	prod_opt_vals_to_prod_opt
products_attributes_download	productsAttrsDownload
customers_basket_attributes	customersBasketAttrs
Column name changes in table configuration_group:	
Old Column Name	New Column Name
configuration_group_description	configurationGroupDesc
Column name changes in table customers_info:	
Old Column Name	New Column Name
customers_info_date_of_last_logon	customersInfoDateLastLogon
customers_info_number_of_logons	customersInfoNumberOfLogon
customers_info_date_account_created	customersInfoDateCreated
customers_info_date_account_last_modified	customersInfoDateModified
Column name changes in table products_options_values_to_products_options:	
Old Column Name	New Column Name
products_options_values_to_products_options_id	prodOptValsToProdOptId

If you study the database upgrade script (notably the one called `upgrade_7.1.1.0_to_7.2.0.0.sql`) for your respective database you will see the SQL that you should run when upgrading from 7.1.1.0 (or 7.1.1.1) to 7.2.0.0.

KonaKart/Torque Programming Changes

If you have used the KonaKart/Torque programming techniques that were given as examples in previous versions you will need to update that code to work with the new KonaKart / Torque 4 framework.

There are just a few simple changes to be aware of:

- Use `com.konakart.db.KKCriteria` instead of `org.apache.torque.util.Criteria` or `com.konakart.bl.KKCriteria`

- Use `com.konakart.db.KKBasePeer` instead of `org.apache.torque.util.BasePeer`
- As a guide, take a look at the example source code that is provided which has been updated for the new version.

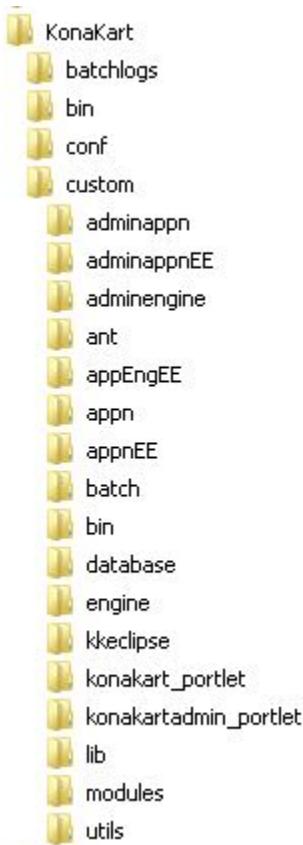
Customizable Source Code

Not all of KonaCart source code is provided in the download kits, only the parts designed to be customized. These include:

- Struts Actions
- Struts Forms
- All JSPs
- All of the Modules (Shipping, Payment, Order Totals, Discount)
- The Client Engine (Enterprise Only)

Source Code Location

Once you have installed KonaCart you will be able to study the customizable source code in the following locations:



You will find all the java sources and associated properties files under the "custom" directory in every download kit from KonaCart version 2.2.1.0 and above. (These source files were available in previous

releases but structured slightly differently). In the future, more directories will be added under the custom directory as more customizable source is made available - so don't be surprised if the directory structure that you see is different to the one on the left.

The "custom" directory is located directly under the installation directory that was selected for KonaKart.

The "custom" directory has various important source directories underneath: "appn"/"appnEE", "engine"/"adminengine" and "modules".

The "appn" directory tree holds the java source for the actions, the forms, the module super-classes and other customisable classes such as CheckoutAction.java.

The "appnEE" directory tree holds the Enterprise-only java source for the actions, the forms, the module super-classes and other customisable classes such as LDAPMgr.java.

The "appEngEE" directory tree holds all of the java classes required to create the KKAppEng "Application Engine" - only available in the Enterprise version.

The "engine" directory tree holds all of the java classes required to create a "custom engine" for KKEngIf.

The "adminengine" directory tree holds all of the java classes required to create a "custom engine" for KKAdminIf.

The "modules" directory tree holds all of the java classes required to implement each of the modules.

The "batch" directory tree holds a selection of example batch jobs that you can modify to suit your own needs.

An ant build file (build.xml) can be found at the root of the custom directory.

The "bin" directory under custom contains a cut-down ant that is sufficient to build all of the custom code - all you need is to set JAVA_HOME to your JDK.

The "lib" directory contains jars for ant builds.

Once you have executed an ant build you will see a few more directories under "custom" which are the products of the build process. For example, you will see classes directories, a new jar directory etc. To remove the files and directories produced by the build you can execute the "clean" target of the ant build, eg:

```
$ bin/kkant clean
```

Building the Customizable Source

An ant file is provided that should make it easy to build the customizable java classes. Use "kkant -p" to see the ant command targets:

```
C:\Program Files\KonaKart\custom>bin\kkant -p
Buildfile: build.xml

Main targets:

build          Compiles all the custom code and creates all the jars
clean          Clears away everything that's created during a build
clean_admin_portlet_war  Clears away the admin portlet war
clean_manifest_file  Clean the MANIFEST file for all jars
clean_portlet_war   Clears away the portlet war
clean_torque_classes  Clears away the generated torque artifacts
clean_wars        Clears away all the WARs and EARs
compile         Compile the customisable application code
compile_admin_portlet_liferay  Compile the Admin portlet utilities for Liferay
compile_adminappn  Compile the customisable admin appn code
compile_adminappnEE  Compile the customisable admin appnEE code
compile_appEngEE   Compile the customisable appEngEE code
compile_appn      Compile the customisable appn code
compile_appnEE    Compile the customisable appnEE code
compile_custom_adminengine  Compile the customisable admin engine code
compile_custom_engine  Compile the customisable engine code
compile_modules   Compile the customisable module code
compile_utils     Compile the customisable utils code
copy_jars        Copy selected custom konakart jars to the lib directories
create_torque_classes  Process the Custom Schema to produce torque classes
debugenv         Debug the environment
enableWebServices  Enable Web Services in deployed server
enable_JSON       Enable JSON in konakart web.xml - EE Only
execute_sql_file Execute the specified SQL file
generate_torque_java  Process the Custom Schema to produce torque java files
make_admin_liferay_portlet_war  Create the konakartadmin portlet war for Liferay
make_ear          Create the konakart EAR
make_eclipse_project  Create an Eclipse Project for Developing the Struts-2 Storefront
make_jar_appEngEE  Create the konakart_app.jar
make_jar_custom    Create the konakart_custom jar
make_jar_customEE  Create the konakart_customEE.jar
make_jar_custom_admin  Create the konakartadmin_custom.jar
make_jar_custom_adminEE  Create the konakartadmin_customEE.jar
make_jar_custom_adminengine  Create the konakartadmin_custom_engine.jar
make_jar_custom_engine  Create the konakart_custom_engine.jar
make_jar_custom_utils  Create the konakart_custom_utils.jar
make_jars          Create the konakart custom jars
make_liferay_portlet_war  Create the konakart portlet war for Liferay
make_manifest_file  Create the MANIFEST file for all jars
make_ordertotal_module_jar  Create the ordertotal module jar
make_other_module_jar  Create the other module jar
make_payment_module_jar  Create the payment module jar
make_shipping_module_jar  Create the shipping module jar
make_torque_jar     Create the konakart_custom_db.jar
make_wars          Create the konakart wars

Default target: build
```

Notice that the default build target compiles all the source files and creates the jars. It doesn't move the jars or create any wars.

If after building the jars you wish to move these to your webapp lib directories you should use the "copy_jars" target.

The "make_wars" target is just a convenient means of creating wars out of the KonaKart webapps. It is not required to be run to build and deploy the custom code so it's not closely-related to the topics discussed on this page.

The "make_*_portlet_war" targets are for creating JSR-286 compliant WARs out of your current KonaKart application. See the section of this documentation regarding portlets for more details.

Here is an example of running the default ant target (Community version with default Struts-2 storefront):

```
C:\Program Files\KonaKart\custom>bin\kkant
Buildfile: build.xml

debugenv:
[echo] custom.home      = C:\Program Files\KonaKart\custom
[echo] java.source      = 1.6
[echo] java.target      = 1.6
[echo] debug_javac      = on
[echo] torque.build.present = ${torque.build.present}
[echo] adminappn.code.present = true
[echo] adminappnEE.code.present = ${adminappnEE.code.present}
[echo] adminengine.code.present = true
[echo] appn.code.present = true
[echo] appnEE.code.present = ${appnEE.code.present}
[echo] engine.code.present = true
[echo] utils.code.present = true
[echo] excludeAXIS      = ${excludeAXIS}
[echo] noClean           = ${noClean}

clean_torque_classes:

clean_portlet_war:
[echo] Cleanup portlet WARS...
[echo] Cleanup portlet classes...
[echo] Cleanup portlet WAR staging area...

clean_admin_portlet_war:
[echo] Cleanup admin portlet WARS...
[echo] Cleanup admin portlet WAR staging area...

clean_wars:
[echo] Cleanup WARS...
[echo] Cleanup EARs...

clean:
[echo] Cleanup...

generate_torque_java:

create_torque_classes:

make_manifest_file:
[echo] Create the MANIFEST.MF file for all jars

compile:
[echo] Compile the customisable application code

compile_appn:
[echo] Copy the properties over to C:\Program Files\KonaKart\custom/appn/classes
[copy] Copying 3 files to C:\Program Files\KonaKart\custom\appn\classes
[echo] Compile the customisable application code
[javac] Compiling 133 source files to C:\Program Files\KonaKart\custom\appn\classes

compile_appnEE:

compile_appEngEE:

compile_adminappn:
[mkdir] Created dir: C:\Program Files\KonaKart\custom\adminappn\classes
[javac] Compiling 10 source files to C:\Program Files\KonaKart\custom\adminappn\classes

compile_adminappnEE:

compile_utils:
[echo] Compile the customisable utils code
[mkdir] Created dir: C:\Program Files\KonaKart\custom\utils\classes
[javac] Compiling 3 source files to C:\Program Files\KonaKart\custom\utils\classes
```

continued...

```

continued from above...

compile_modules:
    [echo] Compile the customisable module code
    [mkdir] Created dir: C:\Program Files\KonaKart\custom\modules\classes
    [javac] Compiling 149 source files to C:\Program Files\KonaKart\custom\modules\classes

make_payment_module_jar:
    [echo] Create the payment module jar
    [mkdir] Created dir: C:\Program Files\KonaKart\custom\jar
    [jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakart_payment_authorizenet.jar

make_payment_module_jar:
    [echo] Create the payment module jar
    [jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakart_payment_elink.jar

-- other payment, order total and shipping module builds removed

compile_custom_engine:
    [echo] Compile the customisable engine code
    [mkdir] Created dir: C:\Program Files\KonaKart\custom\engine\classes
    [javac] Compiling 241 source files to C:\Program Files\KonaKart\custom\engine\classes

compile_custom_adminengine:
    [echo] Compile the customisable admin engine code
    [mkdir] Created dir: C:\Program Files\KonaKart\custom\adminengine\classes
    [javac] Compiling 440 source files to C:\Program Files\KonaKart\custom\adminengine\classes

make_jars:
    [echo] Create the konakart custom jars

make_jar_custom:
    [echo] Create the konakart_custom.jar
    [jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakart_custom.jar

make_jar_customEE:
    make_jar_custom_engine:
        [echo] Create the konakart_custom_engine.jar
        [jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakart_custom_engine.jar

make_jar_appEngEE:
    make_jar_custom_utils:
        [echo] Create the konakart_custom_utils.jar
        [jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakart_custom_utils.jar

make_jar_custom_admin:
    [echo] Create the konakartadmin_custom.jar
    [jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakartadmin_custom.jar

make_jar_custom_adminEE:
    make_jar_custom_adminengine:
        [echo] Create the konakartadmin_custom_engine.jar
        [jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakartadmin_custom_engine.jar

make_torque_jar:
build:
BUILD SUCCESSFUL

```

Note that from KonaKart v6.5.0.0 the default storefront uses Struts-2. Previous versions of the ANT build file required that if you had installed the Struts-1 storefront you needed to add the "-DS1=true" parameter to most of the custom ANT commands. This ANT parameter is no longer supported beyond v6.6.0.0 of KonaKart.

Developing the storefront in Eclipse

The custom ant file contains a target ("make_eclipse_project") that will create an Eclipse project that will allow you to develop the Struts-2 storefront application conveniently in the rich development environment provided by Eclipse.

In earlier versions of the ANT build file a similar "make_eclipse_project_s1" command existed for creating an Eclipse project for working on the Struts-1 storefront. This only worked if you had installed the Struts-1 version of the storefront. This ANT target is no longer included beyond v6.6.0.0 of KonaKart.

Some notes are provided after the Eclipse project is created as follows:

```
C:\Bob\KonaKart\custom>bin\kkant make_eclipse_project
Buildfile: build.xml

make_eclipse_project:
[echo] Create an Eclipse Project for Developing the Struts-2 Storefront
[mkdir] Created dir: C:\Bob\KonaKart\custom\appEngEE\src
[mkdir] Created dir: C:\Bob\KonaKart\custom\kkeclipse\src
[mkdir] Created dir: C:\Bob\KonaKart\custom\kkeclipse\build\classes
[mkdir] Created dir: C:\Bob\KonaKart\custom\kkeclipse\licenses
[mkdir] Created dir: C:\Bob\KonaKart\custom\kkeclipse\resources
[mkdir] Created dir: C:\Bob\KonaKart\custom\kkeclipse\WebContent
[mkdir] Created dir: C:\Bob\KonaKart\custom\kkeclipse\src\com\konakart\bl\modules\others
[mkdir] Created dir: C:\Bob\KonaKart\custom\kkeclipse\src\com\konakartadmin\modules\others
[copy] Copying 2 files to C:\Bob\KonaKart\custom\kkeclipse
[copy] Copying 748 files to C:\Bob\KonaKart\custom\kkeclipse\src
[copy] Copying 18 files to C:\Bob\KonaKart\custom\kkeclipse\resources
[copy] Copying 1 file to C:\Bob\KonaKart\custom\kkeclipse\licenses
[copy] Copying 672 files to C:\Bob\KonaKart\custom\kkeclipse\WebContent
[echo] -----
[echo] Eclipse Project Created on disk - called kkeclipse
[echo] Start Eclipse
[echo] Create a tomcat server (see 'Servers' in Eclipse)
[echo] Move the kkeclipse directory to another location for development if you wish
[echo] Import the kkeclipse project using 'Import existing projects into Workspace'
[echo] Assign a server for kkeclipse to use (define server under Project Properties)
[echo] Edit, Debug and Run the KonaKart Storefront in Eclipse!
[echo] ----

BUILD SUCCESSFUL
```

You can copy the kkeclipse directory to another location before importing into Eclipse if that is more convenient.

You have to create the "Server" that you wish to run the kkeclipse project in. In theory you can choose any server that is supported by Eclipse.

You may have to make one or two modifications to your Eclipse project settings depending on your own environment.

Developing the Product Creation Wizard in Eclipse

The custom ant file contains a target ("make_eclipse_wizard_project") that will create an Eclipse project that will allow you to develop the product creation wizard application conveniently in the rich development environment provided by Eclipse.

Some notes are provided after the Eclipse project is created as follows:

```
C:\Program Files (x86)\KonaKart\custom>ant make_eclipse_wizard_project
Buildfile: build.xml

make_eclipse_wizard_project:
[echo] Create an Eclipse Project for Developing the Product Creation Wizard
[copy] Copying 56 files to C:\Bob\KonaKart\custom\kkeclipseWizard
[copy] Copying 213 files to C:\Bob\KonaKart\custom\kkeclipseWizard\WebContent
[echo] -----
[echo] Eclipse Product Creation Wizard Project Created on disk - called kkeclipseWizard
[echo] Start Eclipse
[echo] Create a tomcat server (see 'Servers' in Eclipse)
[echo] Move the kkeclipseWizard directory to another location for development if you wish
[echo] Import the kkeclipseWizard project using 'Import existing projects into Workspace'
[echo] Assign a server for kkeclipseWizard to use (define server under Project Properties)
[echo] Edit, Debug and Run the KonaKart Product Creation Wizard in Eclipse!
[echo] -----

BUILD SUCCESSFUL
```

You can copy the kkeclipseWizard directory to another location before importing into Eclipse if that is more convenient.

As for the kkeclipse storefront project you have to create the "Server" that you wish to run the kkeclipseWizard project in. In theory you can choose any server that is supported by Eclipse.

You may have to make one or two modifications to your Eclipse project settings depending on your own environment.

Customization of the KonaKart Engines

One of the most important features of KonaKart is the availability of a set of open APIs that allow you to control KonaKart as you require as part of your IT solution. These work fine most of the time but there are occasions, especially with heavily-customized solutions, where you need to modify the behavior of the KonaKart APIs.

KonaKart Customization Framework

A simple framework has been developed that allows you to modify what happens when you call the KonaKart APIs.

It may be instructive to start by explaining the architecture so that you get a clear view of where you should add your customizations.

The interface to the KonaKart application engine is defined in *KKEngIf.java* . This is implemented by *KKEng.java* . A new, custom, implementation of the interface is provided in *KKCustomEng.java* . For every API call, there is a method in *KKCustomEng.java* that in turn calls a method in *KKEng.java* . The call is made by instantiating an instance of a class that has the same name as the method and executing the relevant API call on that class. The source for all of these classes is provided in the download package (from version 2.2.6.0). If you wish to change the behavior of a certain API call, you simply move the relevant source file from *custom/gensrc/com/konakart/app/METHOD.java* to *custom/src/com/konakart/app/METHOD.java* , then modify the code in that java file, to implement the behavior you require for that particular API call.

KonaKart has two major API sets - one for the Application [<http://www.konakart.com/javadoc/server>] , and one for Administration [<http://www.konakart.com/javadoc/admin>] . The paragraph above explained the classes involved in the Application API but there is an equivalent set for the Administration API as follows:

The interface to the KonaKart administration engine is defined in *KKAdminIf.java*. This is implemented by *KKAdmin.java*. A new, custom, implementation of the interface is provided in *KKAdminCustomEng.java*. For every API call, there is a method in *KKAdminCustomEng.java* that in turn calls a method in *KKAdmin.java*. The call is made by instantiating an instance of a class that has the same name as the method and executing the relevant API call on that class. The source for all of these classes is provided in the download package (from version 2.2.6.0). If you wish to change the behavior of a certain API call, you simply move the relevant source file from *custom/gensrc/com/konakartadmin/app/METHOD.java* to *custom/src/com/konakartadmin/app/METHOD.java*, then modify the code in that java file, to implement the behavior you require for that particular API call.

An ANT build file is provided in the standard download kits that help you build your custom code into jars and wars for subsequent deployment.

The next two sections step through the engine customization process in detail. The first illustrates how to add your own code in the *custom()* interface call, and the second shows you how to modify what happens in the *getCustomerForId()* API call.

Adding a New API call

As from KonaKart version 2.2.6.0 there are two custom API calls for both the application APIs and the Admin APIs for this purpose. The default implementations all return null. One of the two calls is called *customSecure* ("Secure" because a sessionId is validated before it is executed), and the other simply called *custom*. An equivalent pair of calls are available on the Admin API.

The calls are defined as follows (you could also find this information in the javadoc):

```
/**
 * A custom interface that you have to provide an implementation
 * for. The default implementation will simply return a null.
 *
 * There are two versions, one that requires a valid sessionId
 * (customSecure) and one that does not (custom).
 *
 * You are free to use the two input String parameters in any way
 * you choose, for example you may wish to use one to indicate which
 * of your custom functions to run, and the other might
 * contain XML to give you a great deal of flexibility - but it's up
 * to you!
 *
 * @param input1
 *          The first input String - can be anything you choose
 * @param input2
 *          The second input String - can be anything you choose
 * @return Returns a String
 * @throws KKException
 */
public String custom(String input1, String input2) throws KKException;
```

Also, the version that checks the session ID:

```

/**
 * A custom interface that you have to provide an implementation
 * for. The default implementation will throw an exception for an
 * invalid sessionId or return a null.
 *
 * There are two versions, one that requires a valid sessionId
 * (customSecure) and one that does not (custom).
 *
 * You are free to use the two input String parameters in any way
 * you choose, for example you may wish to use one to indicate which
 * of your custom functions to run, and the other might contain XML
 * to give you a great deal of flexibility - but it's up to you!
 *
 * @param sessionId
 *          The session id of the logged in user
 * @param input1
 *          The first input String - can be anything you choose
 * @param input2
 *          The second input String - can be anything you choose
 * @return Returns a String
 * @throws KKException
 */

public String customSecure(String sessionId, String input1, String input2)
    throws KKException;

```

So let's work through an example of using the custom API call.

Let's suppose you wanted to create an API call that returns the OrderId of the last order that was processed. (Yes, it's a simple case, but you can make it do whatever you like once you know how to use this mechanism!).

This functionality is best-suited to the Admin API so we'll use that in our example.

1. Move (yes, don't copy but move, because we only want one version of this file to ensure there are no duplicate classes produced) the *Custom.java* file from the *gensrc* directory tree to the *src* directory tree:

```
$ mv custom/adminengine/gensrc/com/konakartadmin/app/Custom.java \
  custom/adminengine/src/com/konakartadmin/app/
```

It is possible to edit these files in the *gensrc* directory tree but this isn't advisable as it will make it harder for you to upgrade to a future version of KonaKart. It's better to separate your customized versions out from the *gensrc* tree so that you can easily see which interfaces you've customized.

2. Implement the code in *Custom.java*

Before you change it, *Custom.java* contains just this:

```

package com.konakartadmin.app;

import com.konakartadmin.bl.KKAdmin;

/**
 * The KonaCart Custom Engine - Custom -
 * Generated by CreateKKAdminCustomEng
 */
public class Custom
{
    KKAdmin kkAdminEng = null;

    /**
     * Constructor
     */
    public Custom(KKAdmin _kkAdminEng)
    {
        kkAdminEng = _kkAdminEng;
    }

    public String custom(
        String input1, String input2) throws KKAdminException
    {
        return kkAdminEng.custom(input1, input2);
    }
}

```

Since the default implementation for these *custom()* and *customSecure()* methods is simply to return null we will need to replace the lines above (marked in bold) with the following code: (full source code for this example is provided in the download kit in *custom/adminengine/examples/com/konakartadmin/app/Custom.java*)

```

public String custom(String input1, String input2)
    throws KKAdminException
{
    /*
     * Run a query that selects the orders_id of the last order
     * processed.
     */

    List<Record> records =
        KKBasePeer.executeQuery( "SELECT max(orders_id) FROM orders" );

    if (records == null)
    {
        // No Orders Found
        return null;
    } else
    {
        return records.get(0).getValue(1).asString();
    }
}

```

3. Compile the new *Custom* class and rebuild the jars

An ANT build file is provided for this purpose - under the *custom* directory in the download kit.

```
C:\Program Files\KonaKart\custom>bin\kkant -p
Buildfile: build.xml

Main targets:

build           Compiles all the custom code and creates all the jars
clean           Clears away everything that's created during a build
clean_admin_portlet_war   Clears away the admin portlet war
clean_manifest_file    Clean the MANIFEST file for all jars
clean_portlet_war     Clears away the portlet war
clean_torque_classes  Clears away the generated torque artifacts
clean_wars          Clears away all the WARs and EARs
compile          Compile the customisable application code
compile_admin_portlet_liferay  Compile the Admin portlet utilities for Liferay
compile_adminappn   Compile the customisable admin appn code
compile_adminappnEE  Compile the customisable admin appnEE code
compile_appEngEE    Compile the customisable appEngEE code
compile_appn        Compile the customisable appn code
compile_appnEE      Compile the customisable appnEE code
compile_custom_adminengine  Compile the customisable admin engine code
compile_custom_engine  Compile the customisable engine code
compile_modules    Compile the customisable module code
compile_utils       Compile the customisable utils code
copy_jars          Copy selected custom konakart jars to the lib directories
create_torque_classes Process the Custom Schema to produce torque classes
debugenv          Debug the environment
enableWebServices  Enable Web Services in deployed server
enable_JSON        Enable JSON in konakart web.xml - EE Only
execute_sql_file   Execute the specified SQL file
generate_torque_java  Process the Custom Schema to produce torque java files
make_admin_liferay_portlet_war  Create the konakartadmin portlet war for Liferay
make_ear            Create the konakart EAR
make_eclipse_project  Create an Eclipse Project for Developing the Struts-2 Storefront
make_jar_appEngEE   Create the konakart_app.jar
make_jar_custom     Create the konakart_custom jar
make_jar_customEE   Create the konakart_customEE.jar
make_jar_custom_admin  Create the konakartadmin_custom.jar
make_jar_custom_adminEE  Create the konakartadmin_customEE.jar
make_jar_custom_adminengine  Create the konakartadmin_custom_engine.jar
make_jar_custom_engine  Create the konakart_custom_engine.jar
make_jar_custom_utils  Create the konakart_custom_utils.jar
make_jars           Create the konakart custom jars
make_liferay_portlet_war  Create the konakart portlet war for Liferay
make_manifest_file  Create the MANIFEST file for all jars
make_ordertotal_module_jar  Create the ordertotal module jar
make_other_module_jar  Create the other module jar
make_payment_module_jar  Create the payment module jar
make_shipping_module_jar  Create the shipping module jar
make_torque_jar      Create the konakart_custom_db.jar
make_wars            Create the konakart wars

Default target: build
```

You can compile and copy the relevant jars in one statement, as follows:

```
C:\Program Files\KonaKart\custom>.\bin\ant build,copy_jars
Buildfile: build.xml

clean_torque_classes:

clean_portlet_war:
    [echo] Cleanup portlet WARS...
    [echo] Cleanup portlet classes...
    [echo] Cleanup portlet WAR staging area...

clean_admin_portlet_war:
    [echo] Cleanup admin portlet WARS...
    [echo] Cleanup admin portlet WAR staging area...

clean_wars:
    [echo] Cleanup WARS...
    [echo] Cleanup EARs...

clean:
    [echo] Cleanup...

generate_torque_java:

create_torque_classes:

make_manifest_file:
    [echo] Create the MANIFEST.MF file for all jars

compile:
    [echo] Compile the customisable application code

compile_appn:
    [echo] Copy the properties over to C:\Program Files\KonaKart\custom\appn\classes
    [copy] Copying 3 files to C:\Program Files\KonaKart\custom\appn\classes
    [echo] Compile the customisable application code
    [javac] Compiling 133 source files to C:\Program Files\KonaKart\custom\appn\classes

compile_appnEE:
    [echo] Compile the customisable appnEE code
    [mkdir] Created dir: C:\Program Files\KonaKart\custom\appnEE\classes
    [javac] Compiling 5 source files to C:\Program Files\KonaKart\custom\appnEE\classes

compile_appEngEE:
    [echo] Compile the customisable appnEE code
    [mkdir] Created dir: C:\Program Files\KonaKart\custom\appEngEE\classes
    [javac] Compiling 38 source files to C:\Program Files\KonaKart\custom\appEngEE\classes

compile_adminappn:
    [mkdir] Created dir: C:\Program Files\KonaKart\custom\adminappn\classes
    [javac] Compiling 10 source files to C:\Program Files\KonaKart\custom\adminappn\classes

compile_adminappnEE:
    [echo] Compile the customisable adminappnEE code
    [mkdir] Created dir: C:\Program Files\KonaKart\custom\adminappnEE\classes
    [javac] Compiling 2 source files to C:\Program Files\KonaKart\custom\adminappnEE\classes

compile_utils:
    [echo] Compile the customisable utils code
    [mkdir] Created dir: C:\Program Files\KonaKart\custom\utils\classes
    [javac] Compiling 3 source files to C:\Program Files\KonaKart\custom\utils\classes

compile_modules:
    [echo] Compile the customisable module code
    [mkdir] Created dir: C:\Program Files\KonaKart\custom\modules\classes
    [javac] Compiling 152 source files to C:\Program Files\KonaKart\custom\modules\classes
```

(- the middle section has been cut out here -)

```

make_other_module_jar:
[echo] Create the other module jar
[jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakart_others_uspsaddrval.jar

compile_custom_engine:
[echo] Compile the customisable engine code
[mkdir] Created dir: C:\Program Files\KonaKart\custom\engine\classes
[javac] Compiling 241 source files to C:\Program Files\KonaKart\custom\engine\classes

compile_custom_adminengine:
[echo] Compile the customisable admin engine code
[mkdir] Created dir: C:\Program Files\KonaKart\custom\adminengine\classes
[javac] Compiling 441 source files to C:\Program Files\KonaKart\custom\adminengine\classes

make_jars:
[echo] Create the konakart custom jars

make_jar_custom:
[echo] Create the konakart_custom.jar
[jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakart_custom.jar

make_jar_customEE:
[echo] Create the konakart_customEE.jar
[jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakart_customEE.jar

make_jar_custom_engine:
[echo] Create the konakart_custom_engine.jar
[jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakart_custom_engine.jar

make_jar_appEngEE:
[echo] Create the konakart_app.jar
[jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakart_app.jar

make_jar_custom_utils:
[echo] Create the konakart_custom_utils.jar
[jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakart_custom_utils.jar

make_jar_custom_admin:
[echo] Create the konakartadmin_custom.jar
[jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakartadmin_custom.jar

make_jar_custom_adminEE:
[echo] Create the konakartadmin_customEE.jar
[jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakartadmin_customEE.jar

make_jar_custom_adminengine:
[echo] Create the konakartadmin_custom_engine.jar
[jar] Building jar: C:\Program Files\KonaKart\custom\jar\konakartadmin_custom_engine.jar

make_torque_jar:

build:

copy_jars:
[echo] Copy the custom jars to their respective lib directories
[copy] Copying 54 files to C:\Program Files\KonaKart\webapps\konakartadmin\WEB-INF\lib
[echo] Copy the custom jars to their respective lib directories
[copy] Copying 55 files to C:\Program Files\KonaKart\webapps\konakart\WEB-INF\lib

BUILD SUCCESSFUL

```

4. Define the Engine Implementation Class

Your custom code will never be executed if we don't define that KonaKart should instantiate the KKAdminCustomEng class as its engine class rather than the default, which is KKAdmin.

A small java program is provided in the download kit (called */java_api_examples/src/com/konakart-admin/apiexamples/CustomExamples.java*) which demonstrates how you would call one version of *custom()* or the other from a java program.

The code in *CustomExamples.java* actually calls *custom()* three times, once with the default *com.konakartadmin.bl.KKAdmin* engine, once with the *com.konakartadmin.app.KKAdminCustomEng* engine and finally using the web services client engine, called *com.konakartadmin.ws.KKWSAdmin*.

When calling using the default *com.konakartadmin.bl.KKAdmin* engine you should see the default response from *custom()* which is a null being returned.

When calling using the *com.konakartadmin.app.KKAdminCustomEng* engine you should see the response from the *custom()* implementation that we coded above (i.e. it should return the highest orderId in the database).

Finally when calling using the *com.konakartadmin.app.KKAdminCustomEng* engine, what you see depends on which engine the web services are configured to use (they could be using the default *com.konakartadmin.bl.KKAdmin* engine or the custom *com.konakartadmin.app.KKAdminCustomEng* engine). Which engine is used in this case is defined in the *konakartadmin.properties* file as follows:

```
# -----
# KonaKart engine class used by the admin web services
# For the default engine use: com.konakartadmin.bl.KKAdmin
# For the custom engine use: com.konakartadmin.app.KKAdminCustomEng

konakart.admin.ws.engine.classname = com.konakartadmin.bl.KKAdmin
```

The above definition is for the Admin App engine class name definition. An equivalent engine class name definition is required for the application web services, and this is defined in *konakart.properties* (you can find this under *webapps\konakart\WEB-INF\classes*)

```
# -----
# KonaKart engine class used by the web services
# For the default engine use: com.konakart.app.KKEng
# For the custom engine use: com.konakart.app.KKCustomEng

konakart.app.ws.engine.classname = com.konakart.app.KKEng
```

Finally, to actually run the *CustomExamples* , change directory to the *java_api_examples* directory where there is another ANT build file for building and running the examples. (You will have to start the KonaKart server before running the *CustomExamples* test if you want a response from the web service call!):

```
C:\Program Files\KonaKart\custom>cd ../java_api_examples

C:\Program Files\KonaKart\java_api_examples>
    ..\custom\bin\ant clean, compile, runCustomExamples
Buildfile: build.xml

clean:
    [echo] Cleanup...

compile:
    [echo] Compile the examples
    [mkdir] Created dir: C:\Program Files\KonaKart\java_api_examples\classes
    [javac] Compiling 11 source files to C:\Program Files\KonaKart\java_api_examples\classes
    [echo] Copy Properties for the examples
    [copy] Copying 1 file to C:\Program Files\KonaKart\java_api_examples\classes

runCustomExamples:
    [java] (?::init::?) Finished Initialising Log4j
    [java] (?::init::?) The configuration file being used is
        /C:/Program%20Files/KonaKart/webapps/konakartadmin/WEB-INF/
            classes/konakartadmin.properties
    [java] (?::init::?) Initialising KKAdmin
    [java] (?::initKonakart::?) KonaKart Admin V2.2.6.0 built 5:03PM 7-Apr-2008 BST
    [java] (?::init::?) Finished Initialising KonaKartAdmin
    [java] (?::init::?) Initialising Torque
    [java] (?::init::?) Initialising KonaKart-Torque for org.apache.torque.adapter.DBMM
    [java] (?::init::?) Finished Initialising Torque
    [java] (?::login::?) User 'admin@konakart.com' has just logged in to the Admin App
    [java]
    [java] Call the custom interface 'Custom' using Engine named
        com.konakartadmin.bl.KKAdmin
    [java] Result was: null
    [java]
    [java] (?::login::?) User 'admin@konakart.com' has just logged in to the Admin App
    [java]
    [java] Call the custom interface 'Custom' using Engine named
        com.konakartadmin.app.KKAdminCustomEng
    [java] Result was: 27
    [java]
    [java] new KKWSAdminSoapBindingStub setup OK to
        http://localhost:8780/konakartadmin/services/KKWSAdmin
    [java]
    [java] Call the custom interface 'Custom' using Engine named
        com.konakartadmin.ws.KKWSAdmin
    [java] Result was: null

BUILD SUCCESSFUL
Total time: 6 seconds
```

Modifying an Existing API call

We will illustrate how this is done with an example using the `getCustomerForId()` Admin API call.

The default implementation of `getCustomerForId()` returns an `AdminCustomer` object for the specified customer Id.

Suppose you want to change the data returned for whatever reason. It may be that you have customer data in another table somewhere that you would like to look up and add to the `AdminCustomer` object whenever `getCustomerForId()` is called (this other table may or may not be a KonaKart table).

Anyway, to keep this really simple so that we don't lose track of the point of this example we will set the String value "CUSTOM SETTING" in the `custom5` attribute of the `AdminCustomer` object that is returned. This does mean we will overwrite the previous `custom5` attribute value if such existed - so you in a proper implementation will have to be aware of which fields are freely available for your use (the custom fields are designed for customers to use in ways that they see fit).

1. Move the *GetCustomerForId.java* file to the *src* directory tree.

We need to modify the default implementation and to do this we move the file from the *gensrc* directory tree to the *src* directory tree, specifically, this means we need to move the *GetCustomerForId.java* from */custom/adminengine/gensrc/com/konakartadmin/app* to */custom/adminengine/src/com/konakartadmin/app*.

2. Add our customizations to *GetCustomerForId.java*

By default the implementation of *GetCustomerForId.java* is as follows:

```
/**
 * The KonaKart Custom Engine - GetCustomerForId
 * Generated by CreateKKAdminCustomEng
 */
public class GetCustomerForId
{
    KKAdmin kkAdminEng = null;

    /**
     * Constructor
     */
    public GetCustomerForId(KKAdmin _kkAdminEng)
    {
        kkAdminEng = _kkAdminEng;
    }

    public AdminCustomer getCustomerForId(
        String sessionId, int customerId) throws KKAdminException
    {
        return kkAdminEng.getCustomerForId(sessionId, customerId);
    }
}
```

We will add our simple changes so that it looks like this: (this file is included in the download kit under the *custom/adminengine/examples* directory structure)

```
public AdminCustomer getCustomerForId
    (String sessionId, int customerId) throws KKAdminException
{
    // Leave the original call to the KonaKart API unchanged
    AdminCustomer myCustomer = kkAdminEng.getCustomerForId(sessionId, customerId);

    // Now add our special value into custom5:
    // (we may have got this value from a database query?)

    myCustomer.setCustom5("CUSTOM SETTING");

    return myCustomer;
}
```

3. Compile the new *GetCustomerForId* class and rebuild the jars:

An ANT build file is provided for this purpose - under the *custom* directory in the download kit.

```
C:\Program Files\KonaKart\custom>bin\kkant -p
Buildfile: build.xml

Main targets:

build          Compiles all the custom code and creates all the jars
clean          Clears away everything that's created during a build
clean_admin_portlet_war  Clears away the admin portlet war
clean_manifest_file  Clean the MANIFEST file for all jars
clean_portlet_war   Clears away the portlet war
clean_torque_classes  Clears away the generated torque artifacts
clean_wars        Clears away all the WARs and EARs
compile         Compile the customisable application code
compile_admin_portlet_liferay  Compile the Admin portlet utilities for Liferay
compile_adminappn  Compile the customisable admin appn code
compile_adminappnEE  Compile the customisable admin appnEE code
compile_appEngEE  Compile the customisable appEngEE code
compile_appn    Compile the customisable appn code
compile_appnEE  Compile the customisable appnEE code
compile_custom_adminengine  Compile the customisable admin engine code
compile_custom_engine  Compile the customisable engine code
compile_modules  Compile the customisable module code
compile_utils    Compile the customisable utils code
copy_jars       Copy selected custom konakart jars to the lib directories
create_torque_classes  Process the Custom Schema to produce torque classes
debugenv        Debug the environment
enableWebServices  Enable Web Services in deployed server
enable_JSON      Enable JSON in konakart web.xml - EE Only
execute_sql_file Execute the specified SQL file
generate_torque_java  Process the Custom Schema to produce torque java files
make_admin_liferay_portlet_war  Create the konakartadmin portlet war for Liferay
make_ear          Create the konakart EAR
make_eclipse_project  Create an Eclipse Project for Developing the Struts-2 Storefront
make_jar_appEngEE  Create the konakart_app.jar
make_jar_custom   Create the konakart_custom jar
make_jar_customEE  Create the konakart_customEE.jar
make_jar_custom_admin  Create the konakartadmin_custom.jar
make_jar_custom_adminEE  Create the konakartadmin_customEE.jar
make_jar_custom_adminengine  Create the konakartadmin_custom_engine.jar
make_jar_custom_engine  Create the konakart_custom_engine.jar
make_jar_custom_utils  Create the konakart_custom_utils.jar
make_jars         Create the konakart custom jars
make_liferay_portlet_war  Create the konakart portlet war for Liferay
make_manifest_file  Create the MANIFEST file for all jars
make_ordertotal_module_jar  Create the ordertotal module jar
make_other_module_jar   Create the other module jar
make_payment_module_jar  Create the payment module jar
make_shipping_module_jar  Create the shipping module jar
make_torque_jar     Create the konakart_custom_db.jar
make_wars          Create the konakart wars

Default target: build
```

You can compile and copy the relevant jars in one statement, as follows:

```
C:\Program Files\KonaKart\custom>.\bin\kkant build,copy_jars
```

4. Define the Engine Implementation Class

Your custom code will never be executed if we don't define that KonaKart should instantiate the *KKAdminCustomEng* class as its engine class rather than the default, which is *KKAdmin*.

A small java program is provided in the download kit (called */java_api_examples/src/com/konakartadmin/apiexamples/GetCustomerExamples.java*) which demonstrates how you would call one version of *getCustomerForId()* or the other from a java program.

The code in *GetCustomerExamples.java* actually calls *getCustomerForId()* three times, once with the default *com.konakartadmin.bl.KKAdmin* engine, once with the *com.konakartadmin.app.KKAdminCustomEng* engine and finally using the web services client engine, called *com.konakartadmin.ws.KKWSAdmin*.

When calling using the default *com.konakartadmin.bl.KKAdmin* engine you should see the default response from *getCustomerForId()* which should show a null being returned for the *custom5* attribute (unless you fill in these values with something else in your own implementation).

When calling using the *com.konakartadmin.app.KKAdminCustomEng* engine you should see the response from the *getCustomerForId()* implementation that we coded above (i.e. it should return "CUSTOM SETTING" in the *custom5* attribute).

Finally when calling using the *com.konakartadmin.app.KKAdminCustomEng* engine, what you see depends on which engine the web services are configured to use (they could be using the default *com.konakartadmin.bl.KKAdmin* engine or the custom *com.konakartadmin.app.KKAdminCustomEng* engine). Which engine is used in this case is defined in the *konakartadmin.properties* file as follows:

```
# -----
# KonaCart engine class used by the admin web services
# For the default engine use: com.konakartadmin.bl.KKAdmin
# For the custom engine use: com.konakartadmin.app.KKAdminCustomEng

konakart.admin.ws.engine.classname = com.konakartadmin.bl.KKAdmin
```

The above definition is for the Admin App engine class name definition. An equivalent engine class name definition is required for the application web services, and this is defined in *konakart.properties* (you can find this under *webapps\konakart\WEB-INF\classes*)

```
# -----
# KonaCart engine class used by the web services
# For the default engine use: com.konakart.app.KKEng
# For the custom engine use: com.konakart.app.KKCustomEng

konakart.app.ws.engine.classname = com.konakart.app.KKEng
```

Finally, to actually run the *GetCustomerExamples*, change directory to the *java_api_examples* directory where there is another ANT build file for building and running the examples. (You will have to start the KonaCart server before running the *CustomExamples* test if you want a response from the web service call!):

```
C:\Program Files\KonaKart\custom>cd ../java_api_examples

C:\Program Files\KonaKart\java_api_examples>
..>custom\bin\ant clean, compile, runGetCustomerExamples
Buildfile: build.xml

clean:
[echo] Cleanup...

compile:
[echo] Compile the examples
[mkdir] Created dir: C:\Program Files\KonaKart\java_api_examples\classes
[javac] Compiling 12 source files to C:\Program Files\KonaKart\java_api_examples\classes
[echo] Copy Properties for the examples
[copy] Copying 1 file to C:\Program Files\KonaKart\java_api_examples\classes

runGetCustomerExamples:
[java] (?::init::?) Finished Initialising Log4j
[java] (?::init::?) The configuration file being used is
      C:/Program%20Files/KonaKart/webapps/
      konakartadmin/WEB-INF/classes/konakartadmin.properties
[java] (?::init::?) Initialising KKAdmin
[java] (?::initKonakart::?) KonaKart Admin V2.2.6.0 built 5:03PM 7-Apr-2008 BST
[java] (?::init::?) Finished Initialising KonaKartAdmin
[java] (?::init::?) Initialising Torque
[java] (?::init::?) Initialising KonaKart-Torque for org.apache.torque.adapter.DBMM
[java] (?::init::?) Finished Initialising Torque
[java] (?::login::?) User 'admin@konakart.com' has just logged in to the Admin App
[java]
[java] Call the custom interface 'Custom' using Engine named com.konakartadmin.bl.KKAdmin
[java] Found: John doe
[java] custom5 = null
[java]
[java] (?::login::?) User 'admin@konakart.com' has just logged in to the Admin App
[java]
[java] Call the custom interface 'Custom' using Engine named
      com.konakartadmin.app.KKAdminCustomEng
[java] Found: John doe
[java] custom5 = CUSTOM SETTING
[java]
[java] new KKWSAdminSoapBindingStub setup OK to
      http://localhost:8780/konakartadmin/services/KKWSAdmin
[java]
[java] Call the custom interface 'Custom' using Engine named com.konakartadmin.ws.KKWSAdmin
[java] Found: John doe
[java] custom5 = null
[java]

BUILD SUCCESSFUL
Total time: 7 seconds
```

Enabling Engine Customizations

So, you've made some engine customizations as described above and you want them to be executed when you run KonaKart.

Assuming you have placed all the newly-built jars in the appropriate lib directories (the ANT build *copy_jars* target will do this for you in a standard tomcat installation) you have to modify some properties files to make KonaKart instantiate the custom engines so that your customizations will actually be executed.

There is a lot of flexibility here as you can define different engines in different places. It's advisable to change only those engine definitions that you need to change however.

- webapps/konakart/WEB-INF/classes/konakart_app.properties

Defines which engine the main Struts Application will use

```
# -----
# KonaKart engine class used by the KonaKart Application users
#
# For the default engine use:      com.konakart.app.KKEng
# For the custom engine use:      com.konakart.app.KKCustomEng
# For the web services engine use: com.konakart.app.KKWSEng

konakart.app.engineclass = com.konakart.app.KKEng
```

- webapps/konakart/WEB-INF/classes/konakart.properties

Defines which engine the Application web services will use

```
# -----
# KonaKart engine class used by the web services
# For the default engine use:      com.konakart.app.KKEng
# For the custom engine use:      com.konakart.app.KKCustomEng

konakart.app.ws.engine.classname = com.konakart.app.KKEng
```

- webapps/konakartadmin/WEB-INF/classes/konakartadmin.properties

Defines which engine the Administration web services will use

```
# -----
# KonaKart engine class used by the admin web services
# For the default engine use: com.konakartadmin.bl.KKAdmin
# For the custom engine use: com.konakartadmin.app.KKAdminCustomEng

konakart.admin.ws.engine.classname = com.konakartadmin.bl.KKAdmin
```

- webapps/konakartadmin/WEB-INF/classes/konakartadmin_gwt.properties

Defines which engine the GWT Admin Application will use

```
# -----
# KonaKart engine class used by the KonaKart Admin Application users
#
# For the default engine use: com.konakartadmin.bl.KKAdmin
# For the custom engine use: com.konakartadmin.app.KKAdminCustomEng
# For the WSs engine use:      com.konakartadmin.ws.KKWSAdmin

konakartadmin.gwt.engineclass = com.konakartadmin.bl.KKAdmin
```

Pluggable Managers

Since version 3.2.0.0, the internal managers of the KonaKart engines are instantiated by name and adhere to interfaces so that they can easily be substituted by alternative implementations. The managers are listed in the konakart and konakartadmin properties files under WEB-INF/classes for both applications.

```

# -----
# KonaKart managers
# When commented out, the default manager is instantiated

konakart.manager.ProductMgr = com.konakart.bl.ProductMgrEE
#konakart.manager.CacheMgr = com.konakart.bl.CacheMgr
#konakart.manager.CurrencyMgr = com.konakart.bl.CurrencyMgr
konakart.manager.SecurityMgr = com.konakart.bl.SecurityMgrEE
#konakart.manager.CategoryMgr = com.konakart.bl.CategoryMgr
#konakart.manager.ConfigurationMgr = com.konakart.bl.ConfigurationMgr
#konakart.manager.CustomerMgr = com.konakart.bl.CustomerMgr
#konakart.manager.EventMgr = com.konakart.bl.EventMgr
#konakart.manager.LanguageMgr = com.konakart.bl.LanguageMgr
konakart.manager.OrderMgr = com.konakart.bl.OrderMgrEE
#konakart.manager.PromotionMgr = com.konakart.bl.PromotionMgr
#konakart.manager.BasketMgr = com.konakart.bl.BasketMgr
#konakart.manager.ShippingMgr = com.konakart.bl.modules.shipping.ShippingMgr
#konakart.manager.PaymentMgr = com.konakart.bl.modules.payment.PaymentMgr
#konakart.manager.OrderTotalMgr = com.konakart.bl.modules.ordertotal.OrderTotalMgr
#konakart.manager.SolrMgr = com.konakart.bl.SolrMgr
#konakart.manager.TaxMgr = com.konakart.bl.TaxMgr
#konakart.manager.EmailMgr = com.konakart.bl.EmailMgr
#konakart.manager.ManufacturerMgr = com.konakart.bl.ManufacturerMgr
#konakart.manager.ReviewMgr = com.konakart.bl.ReviewMgr
#konakart.manager.WishListMgr = com.konakart.bl.WishListMgr
#konakart.manager.MultiStoreMgr = com.konakart.bl.MultiStoreMgr
#konakart.manager.StoreMgr = com.konakart.bl.StoreMgr
#konakart.manager.CookieMgr = com.konakart.bl.CookieMgr
#konakart.manager.AdminEngineMgr = com.konakartadmin.bl.AdminEngineMgr
#konakart.manager.MqMgr = com.konakart.mq.MqMgr
#konakart.manager.CustomerStatsMgr = com.konakart.bl.CustomerStatsMgr
#konakart.manager.CustomerTagMgr = com.konakart.bl.CustomerTagMgr
#konakart.manager.VelocityContextMgr = com.konakart.bl.VelocityContextMgr
#konakart.manager.MiscItemMgr = com.konakart.bl.MiscItemMgr
#konakart.manager.PunchOutMgr = com.konakart.bl.PunchOutMgr

```

As you can see, most of them are normally commented out so that the default manager is used. The above shows a typical set-up for an Enterprise installation.

This pluggable architecture allows the KonaKart professional services team and suitably qualified partners to customize the engine internals in order to satisfy customer requirements that cannot be met by customizing the API calls.

Adding a Shopping Cart via SOAP

This section explains techniques for adding from simple to sophisticated KonaKart functionality to web sites. Code examples are provided to add KonaKart shopping cart functionality to a JSP website.

There's a growing trend for owners of information-only websites to add shopping cart features to capitalize on the extraordinary growth of eCommerce around the globe.

This simple example shows how to add shopping cart functionality to a JSP website using a zero-cost, loosely-coupled, SOAP web services integration with KonaKart.

How To Add a KonaKart Shopping Cart?

So what is the best way to add shopping cart functionality to a website?

One solution would be to code a complete shopping cart solution from scratch and merge this into your website. This has the potential to provide a good solution but would be a huge investment in time if anything but the most trivial cart was required. Before undertaking this development you should be aware of the

extensive functionality provided by KonaKart which would be costly to reproduce. For example, there's customer management, catalogue management, promotions, payment gateways, shipping gateways, and some complicated tax calculations that depend on the location of the shopper and the store.

A quicker and less-risky solution is to integrate with KonaKart, deriving all the benefits of a sophisticated shopping cart solution without requiring a costly software change to your current site. With imaginative use of stylesheets it's straightforward for a creative web-designer to make the two sites have similar look and feel to give the impression of a more-seamless integration.

Why loosely-coupled?

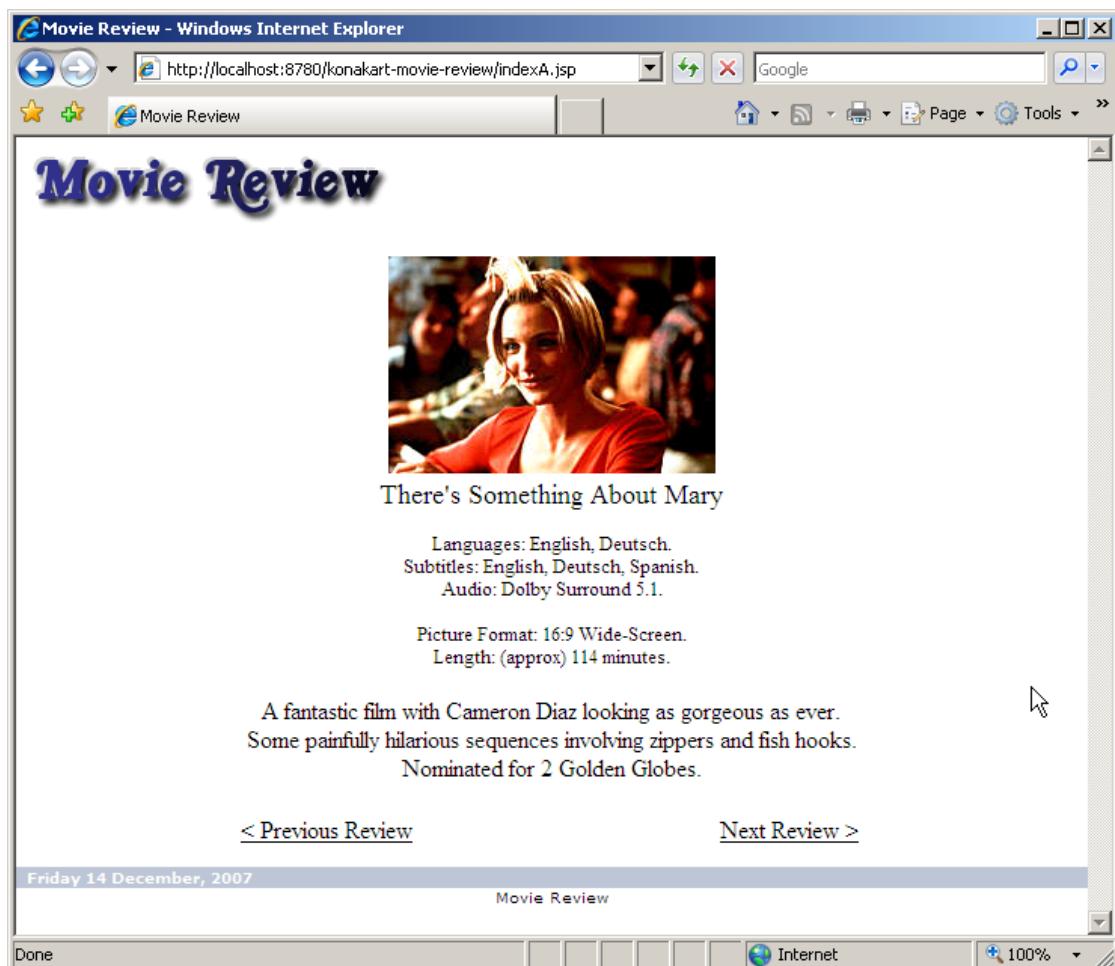
Loose coupling of your current site and the shopping cart site provides a painless migration of an information-only site to an eCommerce-enabled site with minimal effort. You could take this loose-coupling to extremes and introduce a single link from your existing store to your on-line shop and this is a solution many would be content with. However, with very little effort, you can do much better in bringing your site to life with live product information that will encourage your visitors to buy.

The power and simplicity of SOAP interfaces in KonaKart enables this loosely-coupled integration allowing you to maintain your information-only site and online shop completely independently and potentially running with different technologies.

Being SOAP, the simple example described below could have been written in any language that supports SOAP calls, but JSPs have been chosen in this case:

Movie Review Example

Let us imagine, for the sake of this article, that we have a "Movie Review" website that looks like this:



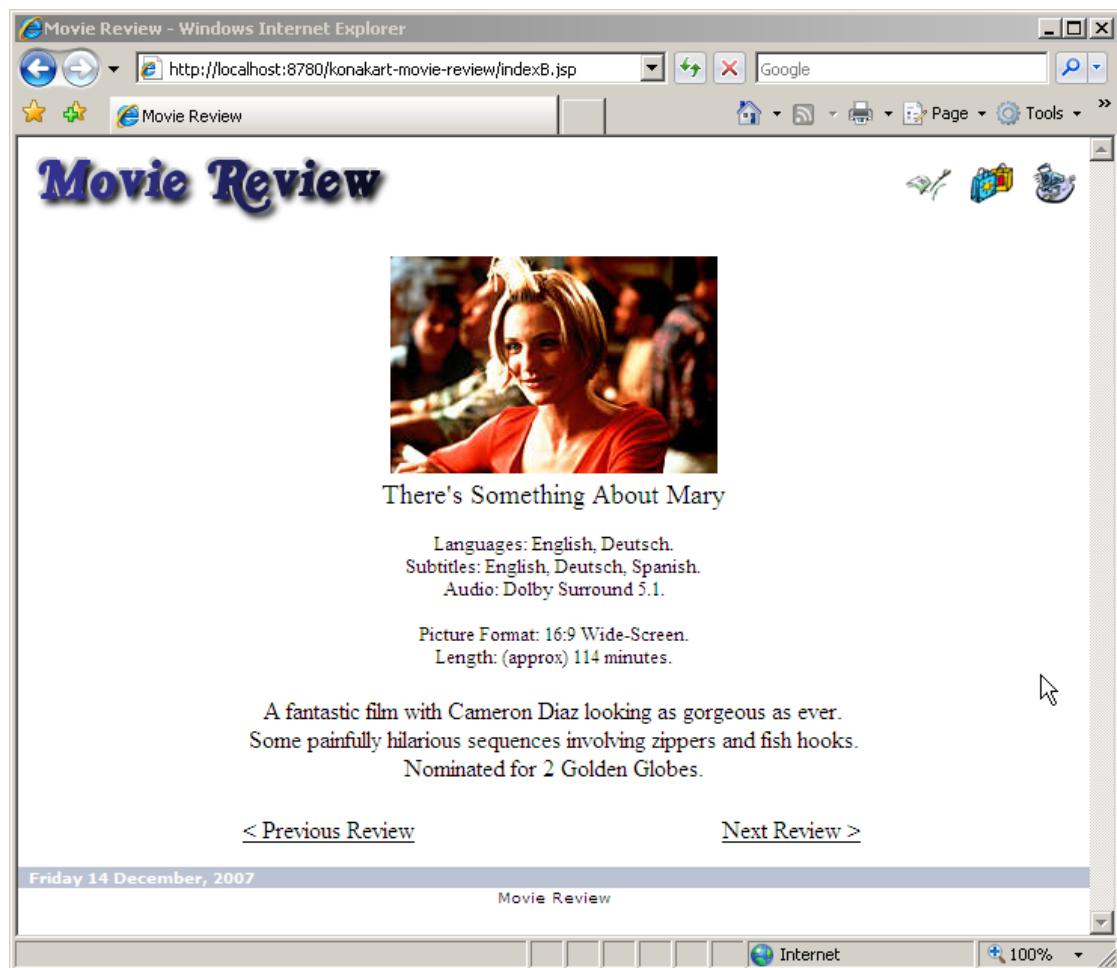
Movie Review Example - Image 1

OK, it's a simple example! Let's say the site contains little more than reviews of the latest movies but has attracted a lively community who might even be contributing with their own reviews.

A natural extension of this website would be to offer DVDs for sale, so rather than build all that shopping cart functionality into Movie Review, we'll integrate with an external site.

The first step would be to add three simple links to the site to take the user directly to his current shopping cart, his account details (to review old orders), and directly to the checkout page. These are simple HTML links which do not require SOAP:

The links were added at the top right:



Movie Review Example - Image 2

Now let's use the power of SOAP to access some information about the DVDs for sale and tempt the visitors of Movie Review to buy. A typical set of data that might be of interest would be a list of new products. We access this in the JSP, in just a few lines of code, as follows:

Please Note!! The code examples here are when you wish to use SOAP stubs generated from the WSDL. In fact we recommend that in a java environment you use the KKWSEng() engine to use the SOAP APIs. For details of using KKWSEng() please refer to the [Using the SOAP Web Service APIs](#) section of this User Guide.

```
// Get the products from the KonaKart engine using a SOAP call

URL endpointUrl = new URL(kkSoapServiceUrl);
KKWSEngIf eng =
    new KKWSEngIfServiceLocator().getKKWebServiceEng(endpointUrl);

DataDescriptor dataDesc = new DataDescriptor();
dataDesc.setOffset(0); // Offset = 0
dataDesc.setLimit(3); // We only want to get back 3 products
dataDesc.setOrderBy(DataDescConstants.ORDER_BY_DATE_ADDED);

Products products = eng.getProductsPerCategory(null,
    /* Data Descriptor defining how many products, the offset
       and the sort order */ dataDesc,
    /* CategoryId. 3 is for DVD Movies */ 3,
    /* searchInSubCats */ true,
    /* The language id. -1 for the default store language */ -1);

Product[] prods = products.getProductArray();
```

Once we have the products back in the SOAP response from the shop we can arrange them how we please on the screen.

Another interesting view we could add is to show a list of best sellers. The code is very similar, but we just use a different SOAP interface:

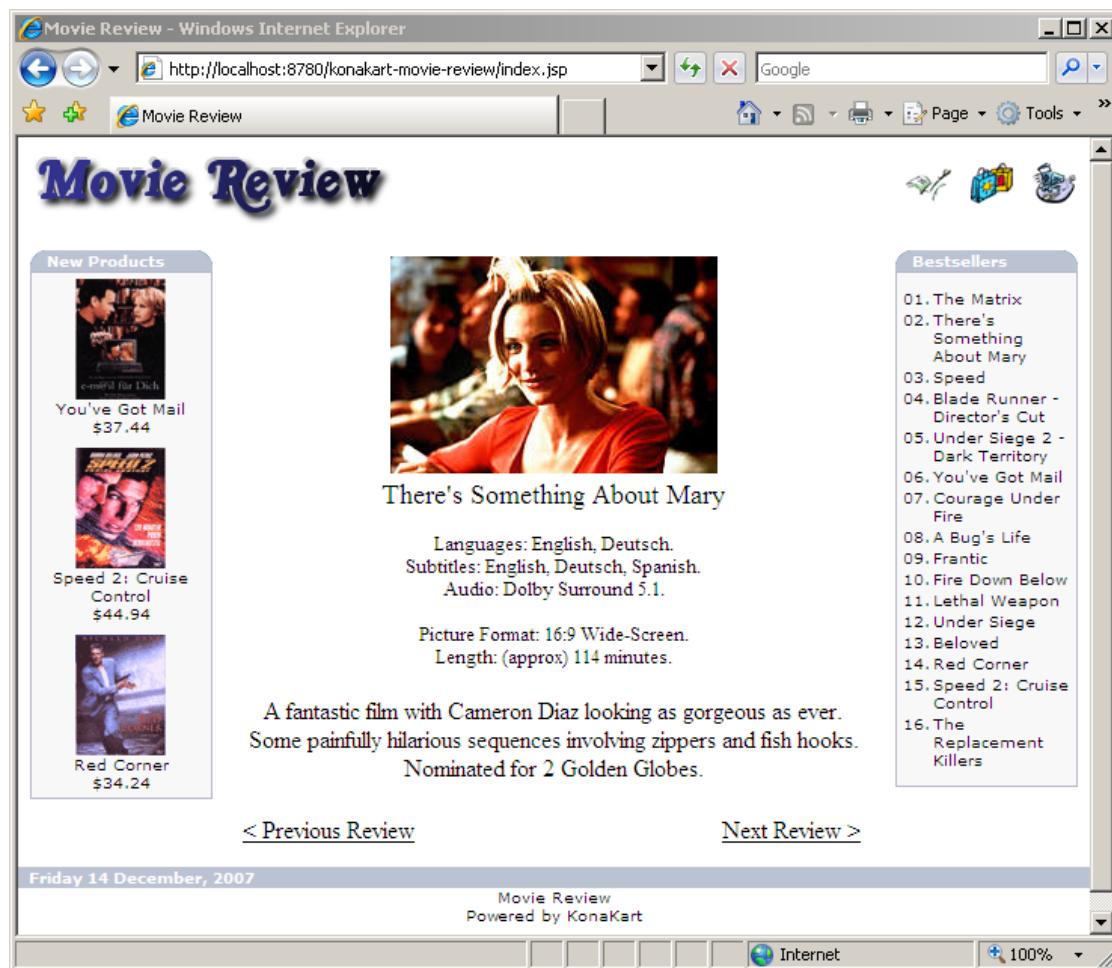
```
// Get the best sellers from the KonaKart engine using a SOAP call

URL endpointUrl = new URL(kkSoapServiceUrl);
KKWSEngIf eng =
    new KKWSEngIfServiceLocator().getKKWebServiceEng(endpointUrl);

DataDescriptor dataDesc = new DataDescriptor();
dataDesc.setOffset(0); // Offset = 0
dataDesc.setLimit(16); // We only want to get back 16 products
dataDesc.setOrderBy(DataDescConstants.ORDER_BY_TIMES_ORDERED);
dataDesc.setOrderBy_1(DataDescConstants.ORDER_BY_NAME_ASCENDING);

Product[] prods = eng.getBestSellers(
    /* Data Descriptor defining how many products, the offset
       and the sort order */ dataDesc,
    /* CategoryId. 3 is for DVD Movies */ 3,
    /* The language id. -1 for the default store language */ -1);
```

Adding these two sets of products to our Movie Review website we now get:



Movie Review Example - Image 3

Every product shown has a link that takes the user directly to the shopping cart, just a few clicks from purchase.

Utilizing the rich set of SOAP APIs provided by KonaKart, the Movie Review website could be improved incrementally with tighter integration.

SOAP client code generation

The simple SOAP calls illustrated above are supported by code that is generated directly from the KonaKart WSDL (at <http://www.konakart.com/konakart/services/KKWebServiceEng?wsdl>)

The steps are simply:

1. Call the Apache AXIS WSDL2Java utility to produce java classes from the WSDL
2. Compile the generated classes
3. Build WAR
4. Deploy to container

Example Source Code

All the source code and ANT build scripts used in this example are available for download in a basic development kit from <http://www.konakart.com/kits/konakart-movie-review-dev-kit.zip> [http://www.konakart.com/kits/konakart-movie-review-dev-kit.zip]

A WAR is also available which contains all the JSPs and everything you need to run the example in a container (such as tomcat) from <http://www.konakart.com/kits/konakart-movie-review-war.zip> [http://www.konakart.com/kits/konakart-movie-review-war.zip]

Feel free to try out the code in either format. It generates code that interfaces with the live KonaKart demo web site so you don't actually have to set up your own independent KonaKart website just to try out this example.

Chapter 19. Reporting

This section describes the KonaKart reporting sub-system. KonaKart uses the popular open source BIRT [<http://www.eclipse.org/birt/>] (<http://www.eclipse.org/birt/>) reporting system but it is loosely-coupled so it is easy to integrate an alternative or additional reporting system if you wish.

The mechanism of Customer events is also explained, and how they can be used to persist important information. This information can be used to generate reports to help you understand how customers are using the KonaKart store and so help you make modifications to improve the overall store performance in terms of usability and sales.

KonaKart Reporting from the Admin App

The reports are accessible from the "Reports" tab of the KonaKart Admin Application. A new browser window will be created that will run the BIRT viewer for the selected report.

Modifying the Reports

You will find the report files under the birtviewer webapp at: *{konakart home}/webapps/birtviewer/reports/stores/store1* By default the reports have the *.rptdesign* file extension although this can be set in the configuration parameters to some other file extension if you wish. For very simple modifications to the reports you can simply edit the report files (they are XML files). You can edit them directly from the Admin Application or using a file editor of your choice. For more complicated modifications or for adding your own new reports, you will have to set up the BIRT/Eclipse report design tools as explained below.

(Note that from version 3.2.0.0 the default reports location has moved to accommodate Multi-Store functionality where each store has its own set of reports. This is why the new location has the store number at the end of the directory path).

(Note that prior to version 2.2.7.0 the "birtviewer" webapp was called "birt-viewer". The change was made due to certain application servers not being able to handle the name "birt-viewer" in all circumstances).

Adding New Reports

KonaKart has been designed to allow the integration of any reporting system. although BIRT [<http://www.eclipse.org/birt/>] has been used in the download package and makes an excellent choice for adding addition reporting functionality.

BIRT [<http://www.eclipse.org/birt/>] is a free Eclipse-based open source reporting system for web applications, especially those based on Java and J2EE. BIRT has two main components: a report designer based on Eclipse, and a runtime component that you can add to your app server. BIRT also offers a charting engine that lets you add charts. Included in the KonaKart download package we provide the "BIRT Viewer" web application that includes the BIRT runtime system which is all you need to produce your reports.

For development of addition reports you are advised to follow the instructions on the BIRT web site for setting up BIRT in Eclipse. Note that currently we are using BIRT 4.3.0 so it is strongly advised that you use that version for new reports to ensure compatibility.

Once you have created your new report you can either upload it using the Admin Application or simply add your report to the directory defined to hold your reports (the directory is a configuration parameter that is set up default to *C:/Program Files/KonaKart/webapps/birtviewer/reports/stores/store1* (therefore,

this will have to be changed if you have not installed KonaKart in the default location on Windows. You can set this value in the Admin Application under the *Reports Configuration* section.)

To make your report appear in the list of reports with a friendly name rather than its filename you have to specify the text to display in the text-property title tag of your BIRT report, for example:

```
<text-property name="title">
    Orders in last 30 Days (Chart Only)
</text-property>
```

Defining a Chart to appear on the Status Page of the Admin App

By default KonaKart is configured to display a plot showing number of orders over the preceding 30 days on the Status page of the Admin Application. What you display in this particular frame is entirely up to you - you can choose another plot, or indeed any page you like that might even be unrelated to KonaKart. You define the URL for the page in the *Reports Configuration* section of the Admin Application.

Reports Configuration

By default KonaKart is configured to use port 8780. If you change this port you will have to modify the "Report Viewer URL" and the "Status Page Report URL" configuration parameters in the *Reports Configuration* section of the Admin Application to reflect the different port. These are set automatically if you installed using the installer so you should only need to do this if you either installed manually using the zip package or changed the KonaKart port number after installation. Here's an example of what you would set these parameters to if you were running KonaKart on port 8080:

KonaKart - Reports Configuration

Defining the Set of Reports Shown in the Admin App

By default KonaKart is configured to search for report files with the *.rptdesign* extension in the *C:/Program Files/KonaKart/webapps/birtviewer/reports/stores/store1* directory. If you use a different file extension for your reports or you have installed KonaKart in any location other than *C:/Program Files/*

KonaKart/ you will have to modify the configuration parameter to suit your installation. You will have to modify the reporting configuration parameters in the Administration Application. You define all of the reporting configuration parameters in the *Reports Configuration* section of the Admin Application.

Accessing the Database in the Reports

KonaKart may be looking for the database connection parameters in the wrong file. Check your *konakart.rptlibrary* file under *webapps/birtviewer/reports/lib/* . Find the beforeOpen section under <data-sources>:

```
<data-sources>
  <oda-data-source
    extensionID="org.eclipse.birt.report.data.oda.jdbc"
    name      ="KonaKart-Database"
    id       ="4">
    <method name="beforeOpen"><![CDATA[
importPackage(Packages.java.util.logging);
importPackage(Packages.com.konakart.reports);

var dbPropsFile =
  "C:/Program Files/KonaKart/webapps/konakartadmin/WEB-INF/classes/konakartadmin.properties";
var dbparams = new GetDbParams(dbPropsFile);

this.setExtensionProperty("odaDriverClass", dbparams.getDbDriver());
this.setExtensionProperty("odaURL", dbparams.getDbUrl());
this.setExtensionProperty("odaUser", dbparams.getDbUser());
this.setExtensionProperty("odaPassword", dbparams.getDbPassword());
```

This is how the reports find out how to connect to the database. The *konakart.rptlibrary* file defines the location of the *konakartadmin.properties* file (or it could be your own file so long as it defines the appropriate database connection parameters). Therefore you must verify that the filename is correct for your environment. The default location is as illustrated above which shows the default location when installed on Windows. (This should be set automatically by the installation wizard but could well be set incorrectly if you used the manual zip-based installation).

Customer Events

Customer Events are events triggered by customer actions and are persisted in the database. Each event contains the id of the customer, the date, the event action (to distinguish between different types of events), the store id and optionally any data related to the event such as the product or customer id.

For example, events allow you to track:

- When the details of a product are being looked at. The id or sku of the product may be saved in the event.
- Whenever a product is added or removed from the cart.
- Whenever a customer visits the store and / or logs in.
- Whenever a customer enters or leaves the checkout process.

The number of events generated by KonaKart is determined by the number of visitors to your store and by how much tracking information you wish to gather. So as not to impact run time performance, the events may be written to a different database rather than to the production database. Also, the KonaKart event manager contains a buffer that receives events from the storefront application and immediately returns control. The buffer is emptied in the background by a separate thread.

Creating Customer Events

The KonaCart application API contains an *insertCustomerEvent(CustomerEventIf event)* API call that may be used to insert events. The Struts action classes of the KonaCart storefront application by default insert a certain number of events if these are enabled in the Configuration >> Customer Details section of the admin app.

The event actions are defined in BaseAction.java.

```
/*
 * Event actions
 */
protected static final int ACTION_NEW_CUSTOMER_VISIT = 1;

protected static final int ACTION_CUSTOMER_LOGIN = 2;

protected static final int ACTION_ENTER_CHECKOUT = 3;

protected static final int ACTION_CONFIRM_ORDER = 4;

protected static final int ACTION_PAYMENT_METHOD_SELECTED = 5;

protected static final int ACTION_REMOVE_FROM_CART = 6;

protected static final int ACTION_PRODUCT_VIEWED = 7;
```

You may define new actions to insert events for your particular requirements. BaseAction.java also contains helper methods (*insertCustomerEvent()*) for inserting the events.

Examples of how to use the *insertCustomerEvent()* method may be found in various struts action classes. For example, to track products being removed from the cart:

```
insertCustomerEvent(kkAppEng, ACTION_REMOVE_FROM_CART, b.getProductId());
```

konakart.properties contains the definition of the database connection for the database where the events are written. By default the installer sets it to the production database.

```
# Enterprise Feature
torque.database.kkstats.adapter      = mysql
torque.dsfactory.kkstats.connection.driver = com.mysql.jdbc.Driver
torque.dsfactory.kkstats.connection.url   = jdbc:mysql://localhost:3306/dbname
                                            ?zeroDateTimeBehavior=convertToNull
torque.dsfactory.kkstats.connection.user  = root
torque.dsfactory.kkstats.connection.password =
```

Chapter 20. Liferay Portal Integration

Introduction

In order to easily integrate the KonaKart Storefront and Admin applications into the popular Liferay portal environment, we supply ANT tasks that automatically generate portlets from the applications which can then be easily imported into Liferay. The storefront application maybe customized within an Eclipse based project to match you requirements before you generate the portlet.

Creation of portlet WAR files

Note that more detailed instructions for Liferay are included in a separate document called *LiferayPortletInstallation.pdf* under the *doc* directory of your KonaKart installation.

In order to create the WAR file that can be loaded into a portal, you must first install KonaKart normally and ensure that it is working correctly.

Navigate to the *KonaKart\custom* directory and run the command `.\bin\kkant -p` to view the ant targets in the supplied build file.

Versions of KonaKart prior to v6.6.0.0 had equivalent ANT build targets for the Struts-1 storefront that was supplied at the time but these are no longer included.

```
C:\Program Files\KonaKart\custom>bin\kkant -p
Buildfile: build.xml

Main targets:

build          Compiles all the custom code and creates all the jars
clean          Clears away everything that's created during a build
clean_admin_portlet_war  Clears away the admin portlet war
clean_manifest_file  Clean the MANIFEST file for all jars
clean_portlet_war   Clears away the portlet war
clean_torque_classes  Clears away the generated torque artifacts
clean_wars        Clears away all the WARs and EARs
compile         Compile the customisable application code
compile_admin_portlet_liferay  Compile the Admin portlet utilities for Liferay
compile_adminappn  Compile the customisable admin appn code
compile_adminappnEE  Compile the customisable admin appnEE code
compile_appEngEE   Compile the customisable appEngEE code
compile_appn      Compile the customisable appn code
compile_appnEE    Compile the customisable appnEE code
compile_custom_adminengine  Compile the customisable admin engine code
compile_custom_engine  Compile the customisable engine code
compile_modules   Compile the customisable module code
compile_utils     Compile the customisable utils code
copy_jars        Copy selected custom konakart jars to the lib directories
create_torque_classes  Process the Custom Schema to produce torque classes
debugenv         Debug the environment
enableWebServices  Enable Web Services in deployed server
enable_JSON       Enable JSON in konakart web.xml - EE Only
execute_sql_file  Execute the specified SQL file
generate_torque_java  Process the Custom Schema to produce torque java files
make_admin_liferay_portlet_war  Create the konakartadmin portlet war for Liferay
make_ear          Create the konakart EAR
make_eclipse_project  Create an Eclipse Project for Developing the Struts-2 Storefront
make_jar_appEngEE  Create the konakart_app.jar
make_jar_custom    Create the konakart_custom jar
make_jar_customEE  Create the konakart_customEE.jar
make_jar_custom_admin  Create the konakartadmin_custom.jar
make_jar_custom_adminEE  Create the konakartadmin_customEE.jar
make_jar_custom_adminengine  Create the konakartadmin_custom_engine.jar
make_jar_custom_engine  Create the konakart_custom_engine.jar
make_jar_custom_utils  Create the konakart_custom_utils.jar
make_jars          Create the konakart custom jars
make_liferay_portlet_war  Create the konakart portlet war for Liferay
make_manifest_file  Create the MANIFEST file for all jars
make_ordertotal_module_jar  Create the ordertotal module jar
make_other_module_jar  Create the other module jar
make_payment_module_jar  Create the payment module jar
make_shipping_module_jar  Create the shipping module jar
make_torque_jar     Create the konakart_custom_db.jar
make_wars          Create the konakart wars

Default target: build
```

The targets that interest us for building the portlet WAR files are:

- make_liferay_portlet_war
- make_admin_liferay_portlet_war

For example, in order to create the portlet for LifeRay, you must run the command as shown below:

```
C:\Program Files\KonaKart\custom>.\bin\kkant make_liferay_portlet_war -DLR606=true
                                         -DnoAXIS=true
Buildfile: build.xml

Buildfile: build.xml

clean_portlet_war:
[echo] Cleanup portlet WARs...
[echo] Cleanup portlet classes...
[echo] Cleanup portlet WAR staging area...

make_liferay_portlet_war:
[mkdir] Created dir:
      C:\Program Files\KonaKart\custom\portlet_war
[mkdir] Created dir:
      C:\Program Files\KonaKart\custom\konakart_portlet\stage\konakart
[mkdir] Created dir:
      C:\Program Files\KonaKart\custom\konakart_portlet\stage\konakart\WEB-INF\jsp
[echo] Lay out the WAR in the staging area
[echo] Copy (almost) the whole konakart webapp to staging area
[copy] Copying 834 files to
      C:\Program Files\KonaKart\custom\konakart_portlet\stage\konakart
[copy] Copied 35 empty directories to 1 empty directory under
      C:\Program Files\KonaKart\custom\konakart_portlet\stage\konakart
[echo] Copy the jars reqd for the portlet to staging area
[copy] Copying 2 files to
      C:\Program Files\KonaKart\custom\konakart_portlet\stage\konakart\WEB-INF\lib
[echo] Copy the config files reqd for the portlet to staging area
[copy] Copying 6 files to
      C:\Program Files\KonaKart\custom\konakart_portlet\stage\konakart\WEB-INF
[echo] Filter the JSPs to staging area for the portlet WAR
[echo] Filter the web.xml to staging area for the portlet WAR
[echo] Filter the struts-config.xml to staging area for the portlet WAR

adjustAppPortletForLR605:

adjustAppPortletForLR606:
[echo] Make adjustments to Application Portlet WAR for Liferay 6.0.6
[echo] Copy the IterateTag patch to staging area
[copy] Copying 1 file to
      C:\Program Files\KonaKart\custom\konakart_portlet\stage\konakart\WEB-INF\classes

adjustAppPortletForLR611:

adjustAppPortletForAXIS:
[echo] Make adjustments to Application Portlet WAR to exclude AXIS

adjustAppPortletForJBoss:

adjustAppPortletForJBossLiferay:

adjustAppPortletForJBossLiferay606:

adjustAppPortletForJBossLiferay611:

adjustAppPortletForKKDemoSite:
[echo] Create portlet konakart.war
[war] Building war: C:\Program Files\KonaKart\custom\portlet_war\konakart.war

BUILD SUCCESSFUL
```

A file called konakart.war is created in the \KonaKart\custom\portlet_war directory. This is the file that you must import into your portal.

Installation Instructions

Liferay

- Using the Liferay plugin installer, install the generated konakart.war file.

- Add a page to Liferay and give it a 1 column layout template.
- Using the "Add Application" tool, add the newly installed KonaKart application to the page.

Liferay for the KonaKart Admin Application

In order for Liferay users to be able to seamlessly log into the Admin App, the KonaKart roles must be edited to add information so that they can be matched with the Liferay roles. The name of the Liferay role must be written into the Custom1 field of the KonaKart role that should be used. For example let's assume that there is a Liferay role called "Administrator" and we want all Liferay users with this Administrator role to log into KonaKart using the KonaKart "SuperUser" role. To do this we must edit the SuperUser role and add the text "Administrator" to the Custom1 field as shown in the diagram below.

Edit Role

Name	Super User
Description	Permission to do everything to all stores
Super User	<input checked="" type="checkbox"/>
Custom1	Administrator
Custom2	
Custom3	
Custom4	
Custom5	
Save Cancel	

Edit Role

A slight complication occurs when running in multi-store mode. In this case the Liferay role name must be in the form role_store. i.e. catalog_store2, order_store1. In the case of catalog_store2, the string "catalog" must be written in the custom1 field of the KonaKart catalog role and the software will ensure that the user will be logged into store2 with that role.

Once the roles have been matched, you can generate and install the WAR using a process similar to the store front application. Note that the WAR for the KonaKart Admin application can also be generated for JBoss by adding "-Djbossliferay=true" as below:

```
C:\KonaKart\custom>.\bin\ant make_admin_liferay_portlet_war -Djbossliferay=true
```

Under some situations you may wish to create your portlets without the ability to provide web services access. This is sometimes required to avoid AXIS version incompatibility problems. To created portlets that don't have the web services add the parameter "-DnoAXIS=true" to your ANT command line.

Now to import the WAR into Liferay:

- Using the Liferay plugin installer, install the generated konakartadmin.war file.
- Add a page to Liferay and give it a 1 column layout template.
- Using the "Add Application" tool, add the newly installed KonaKart Admin application to the page.

There are some additional installation notes for installing KonaKart in Liferay in the document included in the installation package at:

- {KonaKart-Home}/custom/konakartadmin_portlet/liferay/doc/Installation.pdf

Chapter 21. TaxCloud Integration

Introduction

TaxCloud (<http://www.taxcloud.net/>) is a free sales tax service for online retailers. It instantly calculates sales tax for any U.S. address. TaxCloud has been certified by the Streamlined Sales Tax Governing Board, so you can be sure that their tax information is accurate and up-to-date. TaxCloud files your sales tax returns with all Streamlined member states and provides easy-to-understand reports for you to use with the remaining states.

How does the integration work?

A TaxCloud order total module can be enabled to replace the standard Tax order total module. This can be achieved using the Admin App under Modules >> Order Totals. Before saving the change, the module must be configured correctly as can be seen from the diagram below.

TaxCloud Tax		40
<input type="button" value="Install"/>		<input type="button" value="Remove"/>
Display TaxCloud Tax	<input checked="" type="radio"/> true	<input type="radio"/> false
Sort Order	40	
TaxCloud login Id	XXXXXXX	
TaxCloud login key	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	
Store Address1	100 My Store Street	
Store Address2		
Store City	My City	
Store State	My State	
Store Zip	My ZIP	
USPS user id	XXXXXXXXXXXX	
<input type="button" value="Save"/>		<input type="button" value="Cancel"/>

Configure TaxCloud Module

The TaxCloud Login Id and Key can be retrieved after registering with TaxCloud. The address information is the origin shipping address. Finally, the USPS user id (obtained from USPS) is used to do address

verification lookups. If you already verify all customer addresses before saving them, you can edit the TaxCloud module to remove the address verification code so that this id is not required.

Whenever, the TaxCloud module is called by the KonaKart engine, it sends order information to the TaxCloud web service which returns the amount of tax due for each cart item in the order. To be able to calculate the correct tax and to determine whether there are exemptions, the TaxCloud service must know the tax classification for the products in the cart. It figures this out by reading the TIC (Taxability Information Code) applicable for each product. All product categories may have a TIC associated with them. i.e. The TIC for school supplies is "20070" and the TIC for computers is "30100" etc. Within KonaKart you must supply the correct TIC to each defined Tax Class object as shown below. Since every KonaKart product must have a Tax Class, it will automatically pick up the TIC from the Tax Class.

Name	Tax Code	Description
Computers	30100	Computer Equipment
General Goods	0	General Goods and Services

Displaying 1 to 2 (of 2 tax classes)

Title:	General Goods
Tax Code:	0
Description:	General Goods and Services

New Delete Save Cancel

Configure Tax Class

Reconciliation

In order for TaxCloud to prepare accurate monthly reports, KonaKart must confirm the actual amount of sales tax that was collected from the customer. This reconciliation event is done in the OrderIntegrationMgr whenever we detect that the order has been paid for.

By default, the method `manageTax()` in the OrderIntegrationMgr is commented out. If you decide to use TaxCloud, you must un-comment this method and re-compile the OrderIntegrationMgr as explained in another section of this guide (Building Customizable Source).

Points to Note

The TaxCloud order total module will throw an exception and not return an order total if a problem has occurred. It is advisable to modify the storefront application to detect that the Tax order total is missing and

to take action, i.e. This could mean aborting the order or calling the standard Tax order total that calculates tax based on data present in the database.

The TaxCloud APIs use the SSL protocol and require you to have installed a valid SSL certificate. The test certificate present in the download kit is not accepted and an "unable to find valid certification path to requested target" exception is thrown by TaxCloud. For testing, the way to circumvent this problem is to add the TaxCloud server certificate to your trusted Java key store. There are programs freely available to help you to do this. A Google search for "InstallCert" will find many results.

Chapter 22. Thomson Reuters Integration

Introduction

Thomson Reuters (<http://thomsonreuters.com/>) provide a number of sales tax services for online retailers. The module provided in KonaKart uses the Thomson Reuters Web Services to calculate and report tax. It provides a global tax solution.

How does the integration work?

The Thomson Reuters order total module can be enabled to replace the standard Tax order total module. This can be achieved using the Admin App under Modules >> Order Totals. Before saving the changes, the module must be configured correctly as can be seen from the diagram below.

It is important that the address details you provide for the Store are accurate and can be validated successfully by Thomson Reuters. If Thomson Reuters cannot successfully validate the store address it won't be able to calculate tax accurately.

Display Thomson Reuters Tax	<input checked="" type="radio"/> True	<input type="radio"/> False	<input checked="" type="checkbox"/>
Commit Orders to Thomson Reuters	<input type="radio"/> True	<input checked="" type="radio"/> False	<input checked="" type="checkbox"/>
Sort Order	39		<input type="checkbox"/>
Company Role	S		<input type="checkbox"/>
Company Name	Acme Services Ltd		<input type="checkbox"/>
External Company Id	43UK		<input type="checkbox"/>
Seller Role	GB125287732		<input type="checkbox"/>
Tax Service Username	freddie		<input type="checkbox"/>
Tax Service Password	mercury		<input type="checkbox"/>
Add Credentials	<input type="radio"/> True	<input checked="" type="radio"/> False	<input type="checkbox"/>
Part Number Field	Product Custom1	<input type="button" value="▼"/>	<input type="checkbox"/>
Commodity Code Field	Product Custom2	<input type="button" value="▼"/>	<input type="checkbox"/>
Tax Identifier Field	Customer Custom1	<input type="button" value="▼"/>	<input type="checkbox"/>
Seller Street Address	1 Main Street		<input type="checkbox"/>
Seller City	Commerce City		<input type="checkbox"/>
Seller County			<input type="checkbox"/>
Seller State			<input type="checkbox"/>
Seller ZipCode / Postcode	NN3 6 KQ		<input type="checkbox"/>
Seller Country	GB		<input type="checkbox"/>
Ship From Street Address			<input type="checkbox"/>
Ship From City	Runcorn		<input type="checkbox"/>
Ship From County	Northamptonshire		<input type="checkbox"/>
Ship From State			<input type="checkbox"/>
Ship From ZipCode / Postcode	NN14 1EA		<input type="checkbox"/>
Ship From Country	GB		<input type="checkbox"/>
Tax Service Endpoint URL	http://10.128.1.237:8080/sabrix/services/taxcalculationserv		<input type="checkbox"/>
Zone Service Endpoint URL	http://10.128.1.237:8080/sabrix/services/zonelookupserv		<input type="checkbox"/>
Calling System Number	000000000		<input type="checkbox"/>

Configure Thomson Reuters Module

Many of the configuration settings will need to be obtained from Thomson Reuters or the administrator of your Thomson Reuters system as these are closely-related to the settings in your own system.

Whenever, the Thomson Reuters module is called by the KonaKart engine, it sends order information to the Thomson Reuters web service which returns the amount of tax due for each cart item in the order. To be able to calculate the tax correctly the Thomson Reuters service must know the tax classification for the products in the cart and any relevant tax information for the customer (eg. Tax/VAT Code). Each product may have a Part Number and or Commodity Code associated with it which is passed to Thomson Reuters.

Auditing of Order Transactions

In order for Thomson Reuters to prepare accurate sales/tax reports, KonaKart must confirm the actual amount of sales tax that was collected from the customer. This reconciliation event is coded in the OrderIntegrationMgr whenever an order reaches a certain defined state (usually "Payment Received").

By default, the method *manageTax()* in the OrderIntegrationMgr is commented out. If you decide to use Thomson Reuters, you must un-comment this method and re-compile the OrderIntegrationMgr as explained in another section of this guide (Building Customizable Source).

Points to Note

The Thomson Reuters order total module will throw an exception and not return an order total if a problem has occurred. It is advisable to modify the storefront application to detect that the Tax order total is missing and to take action. i.e. This could mean aborting the order or calling the standard Tax order total that calculates tax based on data present in the database.

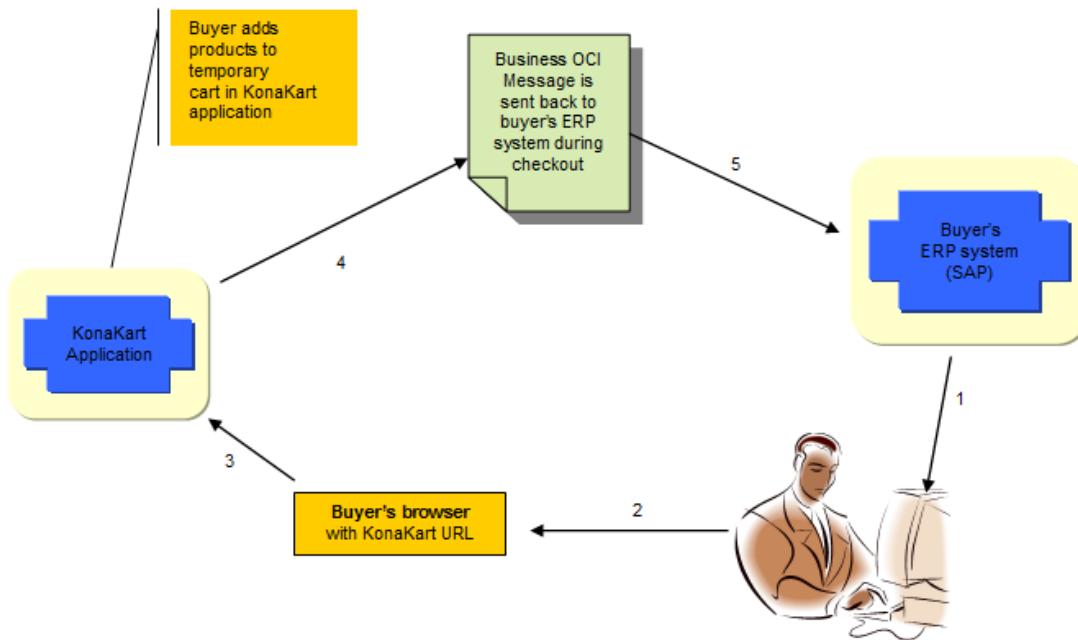
Chapter 23. PunchOut

Introduction

Large companies and government organizations typically use e-procurement systems for purchasing from external suppliers. These systems allow them to easily interface purchasing to other internal systems such as accounting and inventory and they provide benefits like the management of purchasing budgets. PunchOut is the name given to the technical protocol that allows the connection of a product catalog (e.g. a KonaKart store) to an e-procurement system, enabling the system to order from the store.

Currently, KonaKart supports version 4 of the Open Catalog Interface (OCI) standard which is supported by SAP. It generates OCI compliant XML or an HTML message from an array of products that have been added to the cart by the corporate customer and transmits this XML or HTML back to the customer's ERP system.

The process can be described as follows:



PunchOut Process

1. The buyer selects the supplier (KonaKart store) on the procurement application (e.g. SAP).
2. A browser opens for the buyer pointing to the KonaKart application.
3. The PunchOut request is received by the KonaKart application which identifies the B2B buyer, validates his credentials and redirects him to a welcome page. At this point the buyer begins to add products to the cart.
4. When the buyer checks out in the KonaKart application, he is directed back to his procurement application instead of going through the normal KonaKart check out process. An OCI XML or HTML message is sent to the buyer's system, containing information about the list of products he is ordering.

5. The buyer's procurement system receives the checkout products and a standard procurement order is created.

Customization of the PunchOut message

KonaKart contains a PunchOut manager which can be configured in order to customize the XML message or HTML parameters sent by KonaKart. An example of how to do this can be found under the directory *KonaKart\java_api_examples\src\com\konakart\apiexamples* in a file called *MyPunchOutMgr.java*. The methods that must be overridden are called *getData_OCI_XML* and *getData_OCI_HTML*. An example is shown below. In order to use *MyPunchOutMgr* instead of the standard *PunchOutMgr*, you must add it to the *KonaKart\custom\appn\src\com\konakart\bl* directory; compile it and then replace *konakart_custom.jar* with the new jar created in the *KonaKart\custom\jar* directory. The *konakart.properties* file must also be edited to pick up the new manager:

```
konakart.manager.PunchOutMgr = com.konakart.bl.MyPunchOutMgr
```

```
/** 
 * In your own implementation of the PunchOut manager you can override this method to provide
 * your own data for the various tags and tag attributes.
 *
 * @param tagName
 *          The name of the tag as it appears in the XML
 * @param attrName
 *          The name of the attribute as it appears in the XML
 * @param orderProduct
 *          The OrderProduct object. In most cases it will have an attached Product object
 * @param options
 *          The PunchOut options
 * @return Returns the data that will be added to the message. Default values are used if null
 *         is returned
 */
protected String getData_OCI_XML(String tagName, String attrName, OrderProductIf orderProduct,
                                 PunchOutOptionsIf options)
{
    if (tagName.equals("Price") && attrName == null)
    {
        /*
         * Return an empty string to not display the price
         */
        return "";
    } else if (tagName.equals("ItemText") && attrName == null)
    {
        /*
         * The product description isn't returned in the standard implementation
         */
        if (orderProduct.getProduct() != null)
        {
            return "" + orderProduct.getProduct().getDescription() + "";
        }
    } else if (tagName.equals("LeadTime") && attrName == null)
    {
        /*
         * The lead time isn't returned in the standard implementation. It could be saved in one
         * of the Product custom fields.
         */
        if (orderProduct.getProduct() != null)
        {
            return orderProduct.getProduct().getCustom1();
        }
    }

    return null;
}
```

The OCI specification states that many of the possible XML tags are optional. By implementing the `getData_OCI_XML()` method as shown above, you can configure which tags are returned and can alter the data within them.

```
/*
 * In your own implementation of the PunchOut manager you can override this method to customize
 * the data being returned. The parameters are set using code similar to:
 * nvList.add(new NameValue("NEW_ITEM-DESCRIPTION[" + index + "]", op.getName()));
 *
 * @param order
 *          The order
 * @param op
 *          The order product
 * @param nvList
 *          List of NameValue pairs. New data is added to the list.
 * @param index
 *          Index used for creating the name value pairs
 * @param scale
 *          Scale used for formatting the currency
 * @param options
 *          PunchOutOptions
 */
protected void getData_OCI_HTML(OrderIf order, OrderProductIf op, List<NameValue> nvList,
                                int index, int scale, PunchOutOptionsIf options)
{
    // NEW_ITEM-DESCRIPTION
    nvList.add(new NameValue("NEW_ITEM-DESCRIPTION[" + index + "]", op.getName()));

    // NEW_ITEM-VENDORMAT
    if (op.getSku() != null)
    {
        nvList.add(new NameValue("NEW_ITEM-VENDORMAT[" + index + "]", op.getSku()));
    }

    // NEW_ITEM-EXT_PRODUCT_ID
    nvList.add(new NameValue("NEW_ITEM-EXT_PRODUCT_ID[" + index + "]", op.getProductId()));

    // NEW_ITEM-QUANTITY
    nvList.add(new NameValue("NEW_ITEM-QUANTITY[" + index + "]", op.getQuantity()));

    // NEW_ITEM-UNIT
    nvList.add(new NameValue("NEW_ITEM-UNIT[" + index + "]", "EA"));

    // NEW_ITEM-CURRENCY
    nvList.add(new NameValue("NEW_ITEM-CURRENCY[" + index + "]", order.getCurrencyCode()));

    // NEW_ITEM-PRICE Price of an item per price unit
    BigDecimal opPrice = op.getPrice().setScale(scale, BigDecimal.ROUND_HALF_UP);
    if (op.getQuantity() > 1)
    {
        opPrice = opPrice.divide(new BigDecimal(op.getQuantity()), BigDecimal.ROUND_HALF_UP);
    }

    nvList.add(new NameValue("NEW_ITEM-PRICE[" + index + "]", opPrice.toPlainString()));

    // NEW_ITEM-PRICEUNIT
    nvList.add(new NameValue("NEW_ITEM-PRICEUNIT[" + index + "]", "1"));
}
```

This method returns all of the name value pairs for a single Order Product object. It can be customized to return more or less attributes and to use custom attributes when required.

Application PunchOut Code

Two main Struts action classes control the PunchOut process. They are `PunchOutEntryAction.java` and `PunchOutCheckoutAction.java`.

PunchOutEntryAction.java

It is called by the ERP system to start a KonaKart session for the corporate buyer. The customer credentials are validated and if required this action may be used to set the prices for the customer by configuring KKAppEng with a specific catalog id. Details of the PunchOut, such as the return URL are saved in a PunchOut object in KKAppEng.

PunchOutCheckoutAction.java

This action is called when the customer attempts to checkout. It calls the KonaKart engine to get the XML and together with data previously saved in the PunchOut object, it calls CatalogCheckoutPunchout.jsp to post the data back to the corporate buyer's ERP system.