SoftSlate Commerce

User Guide

2.3.5 Edition

SoftSlate, LLC

SoftSlate Commerce: User Guide

SoftSlate, LLC

Table of Contents

List of Tables

List of Examples

Part I. Guide for Administrators

Table of Contents

Chapter 1. Installing SoftSlate Commerce

Overview and System Requirements

The process of installing SoftSlate Commerce consists of first deploying the application to a compatible application server, and then running the Installer tool to configure key settings and initialize the data objects in the database.

The following is a list of system requirements for the server running SoftSlate Commerce. Please make sure you have downloaded and installed the following pieces of software before installing SoftSlate Commerce.

System Requirements for SoftSlate Commerce

- **Java Compiler and Runtime Environment.**

  SoftSlate Commerce has been tested with the Sun JDK, versions 1.4.0, 1.4.2, and 1.5.0, and may work on other versions. The Sun JDK may be downloaded free of charge from http://java.sun.com/j2se/downloads.html [http://java.sun.com/j2se/downloads.html].

- **Java Application Server.**

  SoftSlate Commerce has been tested with Apache Tomcat 4.0, 4.1, 5.0, and 5.5, and may work on other versions. Apache Tomcat may be downloaded free of charge from http://tomcat.apache.org [http://tomcat.apache.org] .

  Users have reported running SoftSlate Commerce sucessfully on Resin, JBoss, and BEA WebLogic. It may work on other compliant J2EE Web Containers as well.

- **Database.**

  SoftSlate Commerce has been tested with MySQL 3.23, 4.0, 4.1, 5.0, and 5.1, and may work on other versions. MySQL may be downloaded free of charge from http://www.mysql.com/downloads [http://www.mysql.com/downloads].

  SoftSlate Commerce has been thoroughly tested with Microsoft SQL Server 2000. Information is available on the Microsoft Web site: http://www.microsoft.com/sql [http://www.microsoft.com/sql].

  SoftSlate Commerce has been tested with PostgreSQL 8.0. PostgreSQL may be downloaded free of charge from: http://www.postgresql.org/download [http://www.postgresql.org/download].

  SoftSlate Commerce has been tested with Oracle 10g Express Edition. Oracle 10g may be downloaded free of charge from: http://www.oracle.com/technology/products/database/xe/index.html [http://www.oracle.com/technology/products/database/xe/index.html] .

  Note that the database used by SoftSlate Commerce need not be installed on the same server that runs the application itself. During the Installer, you can specify the host name of the database server.

- **Operating System.**

  SoftSlate Commerce has been tested on Windows XP, Red Hat Linux, and Solaris 10, but should work well on any operating system supported by Java.

Download SoftSlate Commerce

You should have received a URL, and possibly a user name and password to use to download SoftSlate Commerce. If you don't know where to find the downloads, contact sales or support.

There are three file formats available for you to download. Choose any one of these three files to download; they contain the exact same contents.

- **`softslate-standard-x.x.x.war`.**

  Contains the application in .war (Web application archive) format. Tomcat and other Java application servers can unpackage and deploy .war files automatically. Note: for technical reasons, you cannot use a .war file to install SoftSlate Commerce on BEA WebLogic.

- **`softslate-standard-x.x.x.tar.gz`.**

  Contains the application in tar.gz format. Most commonly used when installing on a Unix or Linux platform.

- **`softslate-standard-x.x.x.zip`.**

  Contains the application in ZIP format. Most commonly used when installing on a Windows platform.

Create an Empty Database

Create an empty database for SoftSlate Commerce to use in MySQL, Microsoft SQL Server 2000, PostgreSQL, or Oracle.

Note

It is possible to install SoftSlate Commerce in an existing database. All of the database tables used by SoftSlate Commerce begin with the prefix `ssc`. If your existing database has no tables that would conflict, you can use it. This may be advantageous in situations where you need to integrate SoftSlate Commerce with other applications.

Example 1.1. Creating an Empty Database in MySQL

1. To, create an empty database in MySQL, log into the MySQL monitor:

```
[root@server root]# mysql -u root -p
Enter password:
```

2. Run the **create database** command. (You may name the database whatever you wish. You will be asked for the name by the Installer tool.)

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 978 to server version: 3.23.58

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>create database softslate;
```

3. If you wish to use the InnoDB engine as the table type in order to support transactions and foreign key constraints, you must update the MySQL table_type system variable before running the installer.

```
mysql>SET GLOBAL table_type=innodb;
```

Example 1.2. Creating an Empty Database in Microsoft SQL Server

You can use the following script to create a database in Microsoft's Query Analyzer application (or a modified form of this script):

```
USE master
GO
create database softslate
on
(
        name='DATA_softslate',
        filename='C:\Program Files\Microsoft SQL Server\MSSQL\Data\DATA_softslate.
        size=2,
        filegrowth=10%
)
log on
(
        name='XLOG_softslate',
        filename='C:\Program Files\Microsoft SQL Server\MSSQL\Data\XLOG_softslate.
        size=1,
        filegrowth=10%
)
GO
```

Important

It's often wise to create a separate database user for each application handled by your database. For example, these commands in MySQL will create a user who only has access to the database

and nothing else:

```
mysql>grant all privileges on softslate.* to 'myuser'@'%' identified by 'mypass
mysql>grant all privileges on softslate.* to 'myuser'@'localhost' identified by
```

Upload and Deploy SoftSlate Commerce

The following example describes in detail how to deploy SoftSlate Commerce on a Linux server running Tomcat. The steps are largely the same for Windows. For other Java application servers, please refer to their documentation for how to install a Web application on their platform.

Example 1.3. Deploying SoftSlate Commerce with Tomcat on Linux

You may either use the `softslate-standard-x.x.x.tar.gz` file or the `softslate-standard-x.x.x.war` file to install SoftSlate Commerce on Linux. (For Windows, it may be easier to use the `.zip` file.) In the first case, you will unpack the application manually. In the second case, you can place the .war file in Tomcat's `webapps` directory and it will unpack the application.

Note

For technical reasons, SoftSlate Commerce will not install correctly on BEA WebLogic when using the `.war` file.

Deploying Using the `.tar.gz` File.

1. If you are using the `.tar.gz` file, change into the directory you wish to install SoftSlate Commerce in, and copy the `.tar.gz` file there. This must be a directory served by Tomcat. For this example, we'll assume we are installing the application in the `/usr/local/apache-tomcat/webapps` directory.

```
[root@server root]# cd /usr/local/apache-tomcat/webapps
[root@server webapps]# cp /root/softslate-standard-x.x.x.tar.gz .
```

2. Untar the `.tar.gz` file. A directory named `softslate` should appear with the contents of the application.

```
[root@server webapps]# tar -zxf softslate-standard-x.x.x.tar.gz
```

3. For the Installer tool to run successfully, the owner of the Tomcat process must have *writable* ac-

cess to the /WEB-INF/conf/keys directory, the /WEB-INF/classes/appSettings.properties file, and the /WEB-INF/classes/hibernate.properties file. If necessary, use **chown** and **chmod** to give the Tomcat user permission.

```
[root@server webapps]# chown <anyuser>.tomcat softslate/WEB-INF/conf/keys
[root@server webapps]# chown <anyuser>.tomcat
                       softslate/WEB-INF/classes/appSettings.properties
[root@server webapps]# chown <anyuser>.tomcat
                       softslate/WEB-INF/classes/hibernate.properties
[root@server webapps]# chmod g+w softslate/WEB-INF/conf/keys
[root@server webapps]# chmod g+w softslate/WEB-INF/classes/appSettings.properti
[root@server webapps]# chmod g+w softslate/WEB-INF/classes/hibernate.properties
```

4. Start the SoftSlate Commerce Web Application. With Tomcat you can use the Manager application to run the start command. For example, if you deployed SoftSlate Commerce under the /softslate context and you have Tomcat running on port 8080, you would use the following URL to start the application:

   ```
   http://<yourhostname>:8080/manager/start?path=/softslate
   ```

Deploying Using the .war File.

1. If you are using the .war file, first *change the name of the .war file to the name you'd like the application to run under* . For example if you'd like the application to use the context /softslate, change the name of the .war file to softslate.war .

   ```
   [root@server root]# mv softslate-standard-x.x.x.war softslate.war
   ```

2. Change into the directory you wish to install SoftSlate Commerce in, and copy the .war file there. This must be a directory served by Tomcat and configured to automatically deploy .war files. For this example, we'll assume we are installing the application in the /usr/local/apache-tomcat/webapps directory.

   ```
   [root@server root]# cd /usr/local/apache-tomcat/webapps
   [root@server webapps]# cp /root/softslate.war .
   ```

   Tomcat should automatically unpack and deploy the application, and a directory named softslate should appear. (The directory name will correspond to whatever name you gave to the .war file.)

   Since Tomcat has deployed the application, there is no need to adjust the permissions or ownership.

Run the SoftSlate Commerce Installer

1. Navigate to the Installer tool using a browser. Visit the `installer` directory under the SoftSlate Commerce installation you've deployed. For example, if you deployed SoftSlate Commerce under the `/softslate` context and you have Tomcat running on port 8080, you would go to this URL: `http://<yourhostname>:8080/softslate/installer`

2. Follow the instructions given by the Installer tool to finish installing SoftSlate Commerce. The Installer tool will ask you how to connect to the database and will ask you to provide key configuration settings. It will then attempt to connect to the database, install all the database objects, and initialize the settings.

3. If you have installed using the `.tar.gz` file, following a successful installation you may wish to change the permissions of the `/WEB-INF/conf/keys` directory, the `/WEB-INF/classes/appSettings.properties` file, and the `/WEB-INF/classes/hibernate.properties` file back for better security.

```
[root@server root]# cd /user/local/apache-tomcat/webapps
[root@server webapps]# chown <restricteduser>.<restricteduser>
                       softslate/WEB-INF/conf/keys
[root@server webapps]# chown <restricteduser>.<restricteduser>
                       softslate/WEB-INF/classes/appSettings.properties
[root@server webapps]# chown <restricteduser>.<restricteduser>
                       softslate/WEB-INF/classes/hibernate.properties
[root@server webapps]# chmod g-w softslate/WEB-INF/conf/keys
[root@server webapps]# chmod g-w softslate/WEB-INF/classes/appSettings.properti
[root@server webapps]# chmod g-w softslate/WEB-INF/classes/hibernate.properties
```

4. After a successful installation, please consider visiting the Getting Started chapter to begin working with SoftSlate Commerce.

Installation Troubleshooting

- **After deploying, the Installer tool does not come up in my browser.**

  There are many reasons why deploying an application might fail. Please refer to the instructions and documentation for the Java application server you are using to troubleshoot any issues. If you are using Tomcat, the Tomcat log file, which is usually located at `<tomcat_home>/logs/catalina.out` or `<tomcat_home>/logs/stdout.log` can provide much-needed clues.

- **The Installer tool fails to connect to the database.**

  If the Installer tool fails to connect to the database, double-check the database name, user name, and password. If you are using MySQL, you may need to associate the database user with your server's host name for the connection to work. (You can refer to the section that describes how to do this under the Installation Procedure for Ensim Servers section.)

- **The Installer tool finished successfully, but I cannot log into the Administrator.**

  It is possible to successfully run the installer after making an error when filling out the "Administrat-

or URL" setting. This is a problem, because the Administrator itself is typically what you would use to make changes to the URL settings.

If logging into the Administrator takes you to the wrong place and you must change the "Administrator URL" setting after finishing the Installer, you must update the URL settings in the database directory. Do this by running the following SQL update commands on the `sscSetting` database table:

```
mysql>use softslate;
mysql>update sscSetting set mediumValue='<base customer URL>'
      where code = 'customerURL';
mysql>update sscSetting set mediumValue='<base secure customer URL>'
      where code = 'customerSecureURL';
mysql>update sscSetting set mediumValue='<base administrator URL>'
      where code = 'administratorURL';
```

For these changes to take effect, you must restart your application server or reload the application.

- **I keep getting `java.lang.ThreadDeath` exceptions running the Installer tool.**

  Some users, typically on systems with limited memory, have reported getting repeated `java.lang.ThreadDeath` exceptions each time the Installer is run, preventing them from completely installing SoftSlate Commerce.

  The first thing to try is simply to restart your application server and try running the Installer again. If it continues to fail, move on the next section, Installing SoftSlate Commerce Manually.

- **The Installer tool finished successfully, but I need to run it a second time.**

  If for whatever reason you need to rerun the Installer tool, you have to do two things: unlock the Installer if it has been locked, and empty the database you are using so the data objects can be reinserted without an error.

  To unlock the Installer tool, edit the `WEB-INF/classes/appSettings.properties` file located under the SoftSlate Commerce installation directory. Look for the line beginning "lockInstaller", and delete that line.

  The easiest way to empty the database is often to simply delete it and recreate it. For example, in MySQL, you could run the following commands to create a new, empty database:

```
mysql>drop database softslate;
mysql>create database softslate;
```

Installing SoftSlate Commerce Manually

It is possible to install SoftSlate Commerce manually without using the Installer tool. Sometimes this is more convenient (or even necessary), especially if you're having trouble with the installer tool. To install manually, follow these steps:

1. Create the empty database to be used for SoftSlate Commerce.

2. Initialize the database objects by running each of the `create*.sql` files followed by the `insert.sql` files under the subdirectories of the `WEB-INF/classes/resources` directory. E.g. `WEB-INF/classes/resources/core/create_mysql.sql`, etc. These files create and insert the database objects needed by SoftSlate Commerce.

3. Run update commands to set the URL settings in the `sscSetting` table according to where your installation lives:

```
mysql>use softslate;
mysql>update sscSetting set mediumValue='<base customer URL>'
      where code = 'customerURL';
mysql>update sscSetting set mediumValue='<base secure customer URL>'
      where code = 'customerSecureURL';
mysql>update sscSetting set mediumValue='<base administrator URL>'
      where code = 'administratorURL';
```

4. Insert a record into the `sscAdministrator` table, so you have an admin login:

```
mysql>insert into sscAdministrator (userName, password, isActive, class)
                values ('myusername', 'mypassword', 1,
                'com.softslate.commerce.businessobjects.administrator.Administr
```

5. Assign the admin user to the superuser role:

```
mysql>insert into sscAdministratorRole (administratorID, roleID, class) values
                'com.softslate.commerce.businessobjects.administrator.Administr
```

6. Update the security setting for admin passwords so you can login without an encrypted password:

```
mysql>update sscSetting set smallValue='noEncryption'
                where code = 'administratorEncryptionType';
```

   You can change this setting back to being encrypted in the Settings -> Security screen, but you have edit the password in the administrator immediately after you do that to encrypt it.

7. To configure the database connection settings manually, edit the `WEB-INF/classes/hibernate.properties` file located under the SoftSlate Commerce installation directory. Add the database name, username, password, and other settings.

8. Edit `WEB-INF/classes/appSettings.properties`. Set the "databaseType" property to one of the following: MySQL, MSSQL, Oracle, or PostgreSQL.

9. Restart your application server.

10. The last thing is to generate the two-way encryption key, if you plan to use two-way encryption for customer password and credit card information (which is the default). The only way to do this cur-

rently is to run the installer. Since the installer creates the key before trying to set up the database, even if you get errors during the installer, it should already have created the key for you. So, run the installer once to create the key.

11. Finally, last but not least, edit `WEB-INF/classes/appSettings.properties` again. Add the following line, which locks the installer from being used:

```
lockInstaller=true
```

Provisioning SoftSlate Commerce with Tomcat on RedHat Linux

This script was written in October, 2007, and it provides a recipie for setting up SoftSlate Commerce in a production environment, on RedHat Linux, using Tomcat and Apache HTTP server. Features of this recipie include installing the Sun JDK, Apache Tomcat, and the Tomcat JK Connector. An unpriviledged user is created to run the Tomcat process, and Tomcat is added to RedHat's initialization system so that it starts automatically on server reboots.

1. Remove the existing Java RPM from the system, if there is one installed. RedHat has shipped with GJC, an open-source implementation of Java which unfortunately is not fully compatible with SoftSlate Commerce.

```
[root@server root]#rpm -qa | grep java
jdk-1.6.0-fcs
[root@server root]#rpm -e jdk-1.6.0-fcs
```

2. Download the Sun JDK, version 5 or 6 from http://java.sun.com/j2se/downloads.html [http://java.sun.com/j2se/downloads.html]. Download the rpm file, which is actually a .bin file. After downloading, you must make it executable and then execute it so you can accept the license agreement. Once you do that it'll extract the .rpm file. Then install the rpm. It should install the JDK into /usr/java.

```
[root@server root]#chmod +x jdk-6-linux-amd64-rpm.bin
[root@server root]#./jdk-6-linux-amd64-rpm.bin
[root@server root]#rpm -i jdk-6-linux-i586.rpm
```

3. Download Apache Tomcat from http://tomcat.apache.org [http://tomcat.apache.org]. Extract it into /usr/local. For convenience, create a symbolic link to it named `/usr/local/tomcat`.

```
[root@server local]#tar -zxf apache-tomcat-5.5.25.tar.gz
```

```
[root@server local]#ln -s apache-tomcat-5.5.25 tomcat
```

4. Create a user and a group on the machine named "tomcat" with no privileges, and change the ownership of certain directories under the Tomcat installation, to the new user and group.

```
[root@server local]#/usr/sbin/useradd tomcat
[root@server local]#/usr/sbin/groupadd tomcat
[root@server local]#chown -R tomcat.tomcat conf logs work temp
```

5. By default Tomcat will initialize all the applications in its webapps directory. On a production installation, this is probably not desireable. To work around this, create an empty directory which we will identify as the webapps directory.

```
[root@server local]#mkdir /usr/local/tomcat/empty
[root@server local]#chown -R tomcat.tomcat /usr/local/tomcat/empty
```

6. Open `/usr/local/tomcat/conf/server.xml` and add the host configuration for the application, within the <Engine> tag. In this example, we are placing SoftSlate Commerce in the web root of the Apache HTTP server:

```
<Host name="mydomain.com" appBase="/usr/local/tomcat/empty" unpackWARs="true"
autoDeploy="true" deployOnStartup="true" xmlValidation="false"
xmlNamespaceAware="false">
<Alias>www.mydomain.com</Alias>
<Context path="" reloadable="true" docBase="/var/www/html" />
</Host>
```

Also in server.xml, change the <Engine> tag's defaultHost attribute to "mydomain.com" (or whatever your domain name is).

7. Install a start/stop script for Tomcat in RedHat's `/etc/init.d` directory. An example of this script is available here: Tomcat Start/Stop script [../tomcat-startstop.txt]. To install the script, simply copy it to `/etc/init.d` and make it executable. Open it to make sure the configurations at the top of the file are correct for your server. To install it into RedHat's initialization system so that Tomcat starts up automatically whenever the server reboots, run these commands:

```
[root@server root]#cd /etc/init.d
[root@server init.d]#/sbin/chkconfig --add tomcat
[root@server init.d]#/sbin/chkconfig tomcat on
```

8. On production systems, it is likely you'll want to run Tomcat under a larger memory (heap) size than the default of 64 Megs. In addition, we must specify the location of the Sun JDK for Tomcat to use it as it starts up. Finally, for the start/stop script to work, we must set an environment variable pointing to a file containing Tomcat's process ID. For these configurations, create a file named `setenv.sh` and place it inside `/usr/local/tomcat/bin`. For a server where you want to specify a heap size of 512 Megs, use the following for the contents of `setenv.sh`:

```
export CATALINA_PID=/usr/local/tomcat/bin/startstop.pid
export JAVA_HOME=/usr/java/default
export JAVA_OPTS="-Xms64M -Xmx512M"
```

9. Test Tomcat at this point by running the start/stop script.

```
[root@server root]#/etc/init.d/tomcat start
```

   Check the Tomcat log file at `/usr/local/tomcat/logs/catalina.out` for any errors or issues. Note, we have not installed SoftSlate Commerce in `/var/www/html` yet, so you should see an error complaining about that.

10. With Tomcat set up, we now need to connect Tomcat with the Apache HTTP server. The idea here is that certain requests for static content such as images should be served by Apache HTTP server. The only requests served by Tomcat will be the dynamic ones that require a Java server application. It is possible to hand off dynamic requests to Tomcat using Apache's Proxy directive. However, the tool of choice for production installations still seems to be the Tomcat JK Connector. So in this example we will install the JK Connector as an Apache module and configure it to redirect dynamic requests to Tomcat. Our first step is to download the Tomcat JK Connector source code from http://tomcat.apache.org/connectors-doc [http://tomcat.apache.org/connectors-doc].

11. Once downloaded, extract and compile the `mod_jk.so` connector from the source code. There are binaries available from Apache but in our experience they have proved incompatible in many cases. Once compiled, copy the `mod_jk.so` library file to `/etc/httpd/modules`.

```
[root@server root]#tar -zxf tomcat-connectors-1.2.25-src.tar.gz
[root@server root]#cd tomcat-connectors-1.2.25-src/native
[root@server root]#./configure --with-apxs=/usr/sbin/apxs
[root@server root]#cp apache-2.0/mod_jk.so /etc/httpd/modules/
```

12. Now, configure Apache HTTP server to load the JK Connector and hand off requests to the JK Connector based on certain URL patterns. Create a file at `/etc/http/conf.d/tomcat.conf`, and use the following for its contents:

```
# Tomcat Connector configuration
LoadModule jk_module modules/mod_jk.so
```

```
JkWorkersFile /etc/httpd/conf.d/workers.properties
JkLogFile /var/log/httpd/jk_module.log
JkLogLevel info
JkMount /*.do worker1
JkMount /*.jsp worker1
JkMount /product/* worker1
JkMount /category/* worker1
JkMount /welcome worker1
JkMount /page/* worker1
JkMount /do/* worker1
JkMount /manager/* worker1
JkMount /manufacturer/* worker1

# Add index.jsp to DirectoryIndex files
DirectoryIndex index.php index.html index.htm index.shtml index.php4 index.php3

# Prevent access to the WEB-INF directory!
<Directory /var/www/html/WEB-INF/>
  AllowOverride None
  Deny from all
</Directory>
```

13. You'll notice the above configuration refers to another configuration file at `/etc/httpd/conf.d/workers.properties`, where the JK Connector's "workers" are defined. Create this file and use the following for its contents to define a worker:

```
# Define 1 real worker using ajp13
# We could define more workers for say, virtual domains. worker.list=worker1, w
worker.list=worker1

# Set properties for the workers (ajp13)

# type: Type of worker
worker.worker1.type=ajp13

# host: Host where is the tomcat worker is listening for ajp13 requests
worker.worker1.host=localhost

# port: Port where the tomcat worker is listening for ajp13 requests
worker.worker1.port=8009
```

14. Now test to make sure the Apache configuration is correct and that it can successfully start up with the JK Connector configuration:

```
[root@server root]#/etc/init.d/httpd configtest
Syntax OK
[root@server root]#/etc/init.d/httpd graceful
```

15. Our last step is to install SoftSlate Commerce itself. Follow the steps in the Installation Instructions of the previous sections to do this. The installation directory for this example is `/var/www/html`, so the *contents* of the SoftSlate Commerce distribution should go directly within `/var/www/html`.

Installation Procedure for Ensim Servers

The following steps are provided if you need to install SoftSlate Commerce on a server running Ensim. These instructions were tested specifically on a server running Ensim Pro for Linux, version 4.0. It may work for other versions as well.

Warning

As of this writing, version 2.x of SoftSlate Commerce has not been tested on Ensim. The following documentation was taken from the installation steps for 1.x. It should apply to version 2.x, but there are no guarantees.

Example 1.4. Installing SoftSlate Commerce on an Ensim Server

1. **Provision the Tomcat Service for the Site You're Setting Up.**

Note

You must set up the site `.policy.custom` file *before* provisioning Tomcat for the site. Otherwise, the site's `.policy.custom` file will not take effect.

a. Log on to the server through ssh.

b. Change to the Tomcat site policies configuration directory:

```
[root@server root]# cd /var/tomcat4/conf/sites.policies.d
```

c. Create a file in this directory named `site15.policy.custom`, whose contents are the following. (Replace "site15" with the Site ID of the site you are setting up.)

```
permission java.security.AllPermission;
```

d. Log in to Ensim's Appliance Administrator control panel.

e. Click on the "Site Manager" link and find the site you are installing SoftSlate Commerce on.

f. Click the pencil icon next to the site name.

g. The site must have Tomcat 4 provisioned for it. If it is not already checked, mark the checkbox next to Tomcat 4, which is under the "Web Server" heading.

h. Tomcat 4 will not be assigned to the site if its "Security Level" is set to "High". If the security level is set to "High", you must change it to "3.1 Compatibility".

2. **Create an Empty Database for SoftSlate Commerce.**

a. Log in to Ensim's Appliance Administrator control panel.

b. Click on the "Site Manager" link and find the site you are installing SoftSlate Commerce on.

c. Click the pencil icon next to the site name.

d. The site must have at least one available MySQL database. Under the "Database Server" heading, make sure the checkbox next to "MySQL" is checked and that the site has at least one database.

e. Click "Save" to record your changes.

f. Navigate back to the "Site Manager" list.

g. Click on the name of the site you are setting up, and click yes to become the Site Administrator.

h. From the Site Administrator control panel, click on Services.

i. Click on the pencil icon next to MySQL.

j. Click on the "Create Database" link.

k. Create a database with "softslate" as the suffix. E.g. `www_sitename_com_-_softslate`.

l. If you have made a change to the site's database password, you may need to log back into Ensim's Appliance Administrator control panel and restart MySQL from the Services menu for the changes to take effect.

3. **In MySQL, Associate the Database User with the Proper Host.**

Note

On some servers, this step may not be necessary. If SoftSlate Commerce's Installer is having trouble making a connection to the database, you may need to perform these steps. MySQL must have an entry in the `user` table that associates the user of the site's database with the host name of your server. In some cases, this is taken care of when you set up a site.

a. Log on to the server through ssh and look up the hostname of your server by running the **hostname** command:

```
[root@server root]# hostname
server.domainname.com
[root@server root]#
```

b. Log in to Ensim's Appliance Administrator control panel.

c. Navigate to "Services" and click on the pencil icon next to MySQL

d. Click the link to "MySQL Administration Tool"

e. Log into MySQL with the root username and password.

f. In phpMyAdmin, select mysql under the "Databases..." menu.

g. Click on the link to the `user` table.

h. Click on the "Insert" tab to add a record to the `user` table.

i. For Host use the hostname of your server from step 1.

j. For User use the MySQL user corresponding to the site you are setting up. (Click on the Browse tab if you need to find it.)

k. For Password, select "PASSWORD" in the drop down menu under the Function column. Enter in the user's password for database access in the Password field.

l. Leave all the radio buttons marked as "N" for No.

m. Click the Go button at the bottom of the page to add the record.

n. Navigate back to the Ensim control panel home, and go back to the "Services" menu.

o. Click the restart icon next to MySQL to restart the database and load your changes.

Alternatively, you may add the user record through the MySQL command line administration interface, and restart MySQL from the command line:

```
[root@server root]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 978 to server version: 3.23.58

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> insert into user values('server.domainname.com', 'sitedbusername',
password('sitedbuserpassword'), 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N
'N', 'N', 'N', 'N');
mysql> \q
Bye
[root@server root]# /etc/init.d/mysqld restart
```

4. **Add Mappings to the Site's Apache Configuration.**

Note

By default, Ensim will redirect URL's that match *.jsp to Tomcat to process. Since SoftSlate Commerce uses *.do and /do/* as well, you must add these mappings to the Site's Apache configuration.

a. Log on to the server through ssh.

b. Change to the Site's Apache configuration directory. Assuming the Site's ID is 15, change to the following directory:

```
[root@server root]# cd /etc/httpd/conf/site15
```

c. Create a file in this directory named `softslate` , whose contents are the following. (Replace "site15" with the Site ID of the site you are setting up.)

```
<Directory /home/virtual/site15/fst/var/www/html/softslate/WEB-INF/>
  AllowOverride None
  Deny from all
</Directory>

<IfModule mod_jk.c>
  JkMount /*.do  ajp13
  JkMount /softslate/do/*  ajp13
</IfModule>
```

d. Make sure to remove any extraneous files in the directory. (E.g., if you used Emacs, be sure to remove the `softslate~` file, if it exists. Apache will attempt to load all the files in the directory.)

e. Restart Apache for the new configuration to take effect:

```
[root@server root]# /etc/init.d/httpd restart
```

5. **Upload and Unpack SoftSlate Commerce.**

a. Upload the SoftSlate Commerce `.tar.gz` file to the server.

b. Log on to the server through ssh.

c. Change into the Site's `html` directory, and copy the `.tar.gz` file there. (Replace "site15" with the Site ID of the site you are setting up.)

```
[root@server root]# cd /home/virtual/site15/fst/var/www/html/
[root@server html]# cp /root/softslate-standard-x.x.x.tar.gz .
```

d. Untar the `.tar.gz` file:

```
[root@server html]# tar -zxf softslate-standard-x.x.x.tar.gz
```

e. For the Installer tool to run successfully, the owner of the Tomcat process must have *writable* access to the `/WEB-INF/conf/keys` directory and the `/WEB-INF/classes/appSettings.properties` file. Use **chown** and **chmod** to give the `tomcat4` user permission. (Replace "admin15" with the Site ID of the site you are setting up.)

```
[root@server html]# chown admin15.tomcat4 softslate/WEB-INF/conf/keys
[root@server html]# chown admin15.tomcat4 softslate/WEB-INF/classes/appSetti
[root@server html]# chown admin15.tomcat4 softslate/WEB-INF/classes/hibnerat
[root@server html]# chmod g+w softslate/WEB-INF/conf/keys
[root@server html]# chmod g+w softslate/WEB-INF/classes/appSettings.properti
[root@server html]# chmod g+w softslate/WEB-INF/classes/hibernate.properties
```

6. **Run the SoftSlate Commerce Installer.**

a. Visit `http://sitedomainname/softslate/installer` in a browser and follow the instructions to install SoftSlate Commerce.

b. If the application does not come up, use the Tomcat log file at `/var/tomcat4/logs/catalina.out` to troubleshoot the issues.

c. If the Installer tool fails to connect to the database, double-check the database name, user name, and password. You may need to associate the database user with your server's host name for the connection to work (see above).

d. Following a successful installation, you should change the permissions of the `/WEB-INF/conf/keys` directory, the `/WEB-INF/classes/appSettings.properties` file, and the `/WEB-INF/classes/hibernate.properties` file back for better security. (Replace "site15" and "admin15" with the Site ID of the site you are setting up.)

```
[root@server root]# cd /home/virtual/site15/fst/var/www/html/
[root@server html]# chown admin15.admin15 softslate/WEB-INF/conf/keys
[root@server html]# chown admin15.admin15 softslate/WEB-INF/classes/appSetti
[root@server html]# chown admin15.admin15 softslate/WEB-INF/classes/hibernat
[root@server html]# chmod g-w softslate/WEB-INF/conf/keys
```

```
[root@server html]# chmod g-w softslate/WEB-INF/classes/appSettings.properti
[root@server html]# chmod g-w softslate/WEB-INF/classes/hibernate.properties
```

Chapter 2. Getting Started

Adding Your First Product

When you first install SoftSlate Commerce, it will be fully functional, but there will be no products in the store for your customers to buy. Follow these basic steps to add your first product and category to the store so you can start selling.

Example 2.1. Adding a Product

1. Log into the SoftSlate Commerce administrator using the user name and password you created in the Installer tool. By default the administrator application is located at `/softslate/administrator`
2. Once logged in, navigate to `Products -> Products`. Click the "Add New Record" button on the right hand side of the screen to go to the add new product form.
3. After adding the product, create a category to place the product in, so it will show up in the store's category tree. Navigate to `Products -> Categories`, and click the "Add New Record" button to add a category to the store.
4. Now assign your new product to the category you created:
    a. Navigate to `Products -> Categories` .
    b. You'll see your new category in the table in the center of the screen. Click the "Products" link next to it to view the products assigned and not assigned to the category.
    c. You should see your new product in the table in the center of the screen. Click the "On" button next to "Edit Mode"
    d. Check the checkbox under the "Assigned" column next to the product.
    e. Click "Update/Delete" to save the change.

Placing a Test Order

Now that you've added a product to your store, you are open for business. Try placing a test order to make sure everything's working right.

Example 2.2. Placing an Order

1. Navigate to the welcome screen of SoftSlate Commerce. By default the welcome screen is located at the root of the application, or `/softslate`. You should see your new category appearing in the "Browse" box on the left hand side.
2. Click on the name of the category. This should take you to a page where your new product is listed. Click on the product's name. You should arrive on the product page, where you can add the product

to your cart.

3. Click the "Add to Cart" button to add the product to your cart, and then click the "Checkout" link to begin the checkout process.

4. Proceed through the checkout process. Create a test customer account, and enter a test address. When you arrive at the payment screen, use "Visa" as the credit card name, and "4111111111111111" as the credit card number. Finally, confirm your order to finalize it.

Viewing Your Test Order

After placing a test order you can go back into the administrator application to view the order and its details.

Example 2.3. Viewing an Order

1. Log into the SoftSlate Commerce administrator using the user name and password you created in the Installer tool. By default the administrator application is located at `/softslate/administrator`

2. Once logged in, navigate to `Orders and Customers -> Orders`. You should see the test order you placed at the center of the screen. Click the "Details" link next to the order to view the billing information for the order.

3. From the order details screen, you can click the "Deliveries" link to view the order's delivery address and information, and from there you can follow links to view the items that were ordered. Click the "Payments" link to view the order's payment information. And click "Display Invoice" to view the invoice for the order, with all of the items and delivery information on one screen.

Congratulations! You've added your first product to your store, and placed your store's first order.

Chapter 3. Important Settings

Administrators can configure a large number of settings that control various aspects of SoftSlate Commerce, under the Settings menu of the Administrator application. The settings are organized for convenience into various categories, such as Store, System, Logic, Display, and so on.

Each individual setting under these categories is accompanied by a description explaining what its purpose is. In the vast majority of cases, the descriptions should be self-explanatory. But for some key settings, we've written the following sections to provide additional information on how to use them to the best effecct.

Store Settings

On the `Settings -> Store` screen, there are several settings that are important for various store operations:

- Store Name and Address: The store's name as defined on the `Settings -> Store` screen will appear in the title bar of all pages (as part of the HTML head's title tag). The address appears to customers as the address on the invoice.
- Store Email Address: It's necessary to define an email address for store. Not doing so will prevent order confirmation and notification emails from being sent. The email address is used as the "from" address for all emails produced by the store. Give some thought as to what the address should be.
- SMTP Server: Likewise, the SMTP server is a necessary setting for the emails produced by the store to go out smoothly.
- Lost Password Email Subject, Text, and HTML templates: These fields define the text and subject of the "lost password" emails sent to customers who have forgotten their password and have filled out the lost password form. >The lost password templates make use of placeholders to indicate where within the template dynamic data, such as the user's password should appear. For a list of all of the placeholders available for these and other settings in the Administrator, please refer to the Template Placeholders section.

System Settings

The following settings are found on the `Settings -> System` screen.

- Store Currency Code: Here you must select the three-letter international currency code that the store is offering its products in.

  SoftSlate Commerce supports international currency codes in the sense that you can select any ISO 4217 currency code for this setting. The store will look at each user's browser setting to determine the user's locale or origin. It will use the individual's locale to determine how to *format* the prices displayed throughout the store. For example, if your store's currency code is USD, for US dollars, a price would be displayed as `$12.50` for a user whose browser settings indicated he or she is from the US, whereas it would be displayed as `12.50 USD` if their browser indicates he or she is from another country.

  There is currently no support in SoftSlate Commerce for selling the same products in multiple currencies and translating their prices through an exchange rate system.

- Built-In Categories: This field allows you to define a list of "built-in" categories, which means the categories and their products are available for display on any page of that application. You may enter a comma-separated list of containing the values of the categories' code fields to define the built-in categories.

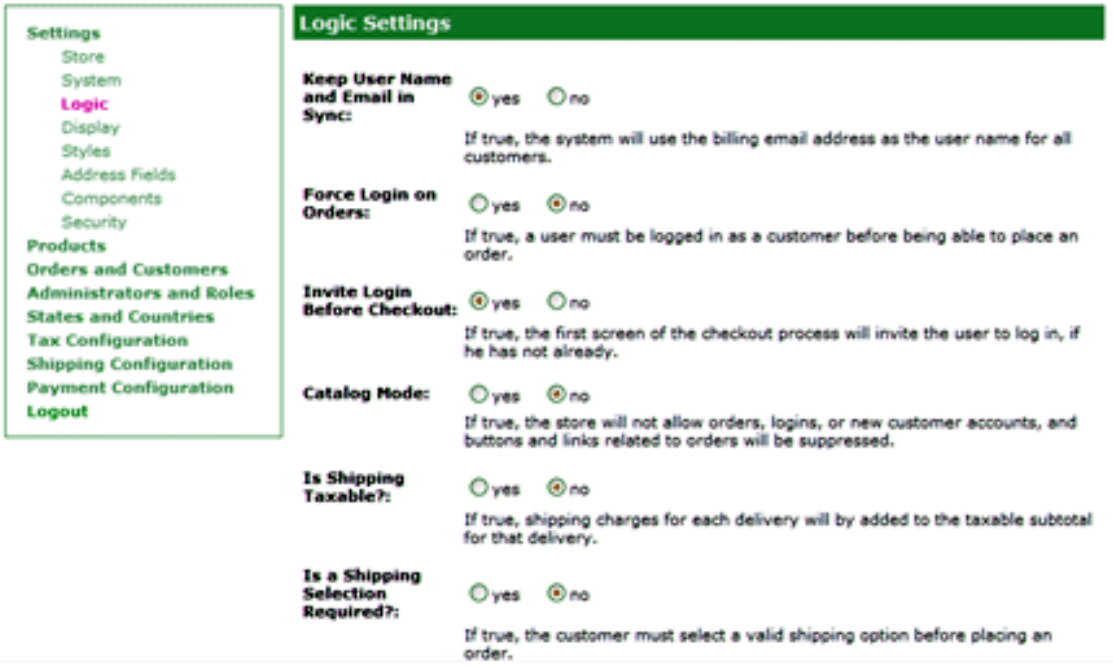
Note

When you update the Built-In Categories setting, it does not automatically reload the built-in categories you enter. To trigger a reload, simply navigate to the Details screen for one of the categories and save the details without making changes.

- URL Path to Images and System Path to Images: These two settings affect the behavior of the image uploads on the product and category details page in the Administrator. You must define these fields correctly for the image uploads to work correctly. Specifically, the "System Path to Image" must represent a full, valid path to a directory on the server where the images should be stored. The directory must be writable by the process owning the Java application server you are running. The "URL Path to Images" is the path browsers should follow under the web root to find the same directory.

Shipping Selection Required Setting

On the `Settings -> Logic` screen, a common change you will want to make is to change the "Is a Shipping Selection Required?" setting from no to yes.



Settings -> Logic Screen

One of the first things you will probably want to do is define at least one shipping method under the `Shipping Configuration -> Shipping Methods` screen. However, when you have done this your new shipping method will not show up during checkout to users unless you have also switched the "Is a Shipping Selection Required?" setting from no to yes.

SoftSlate Commerce is distributed with this setting turned off so you can place test orders immediately

after installation, without having to define a shipping method.

Security Code Required Setting

On the `Settings -> Logic` screen, please take special note of the setting named "Is a Security Code Required for Credit Cards?".

This is a global setting that all of SoftSlate Commerce's payment processors that ask for credit card information, including the Basic Payment Processor and Payflow Link and Payflow Pro processors, use when validating the user's submission of credit card information.

If it's set to "yes", the user must submit a security code with the credit card number. The security code is the three- or four-digit number on the back of the credit card. It is also variously known as the CVV2, CVC2, or CID number.

Default Layout Setting

An important setting for controlling multiple custom layouts is the "Default Layout" setting in the `Settings -> Display` screen of the administrator.

This setting tells SoftSlate Commerce which layout to give users by default, if the layout is not specified on the URL. Initially, it is set to "custom", meaning that fresh visitors coming to site will get the templates in the `/WEB-INF/layouts/custom` directory, if they exist. However, since the `custom` directory is empty when SoftSlate Commerce is first initialized, in effect visitors see the templates in the `default` directory when the store is first installed.

To clarify, here are the steps SoftSlate Commerce takes to determine which JSP template to use for a given user:

1. SoftSlate Commerce first looks for a request parameter on the URL or posted in a form named *`layout`*. If this parameter exists, its value is placed in the user's session and is carried with the user until the session expires.

2. If the user's *`layout`* has been set, SoftSlate Commerce looks for the JSP template under a directory of the same name, inside the `/WEB-INF/layouts` directory. In other words, if the user's *`layout`* has been set to `special`, the system uses the JSP templates inside the `/WEB-INF/layouts/special` directory.

3. If a given template does not exist under the directory defined by the *`layout`* parameter, the system next looks inside a directory of the same name as the Default Layout setting defined in the `Settings -> Display` screen. By default, this would be the `/WEB-INF/layouts/custom` directory.

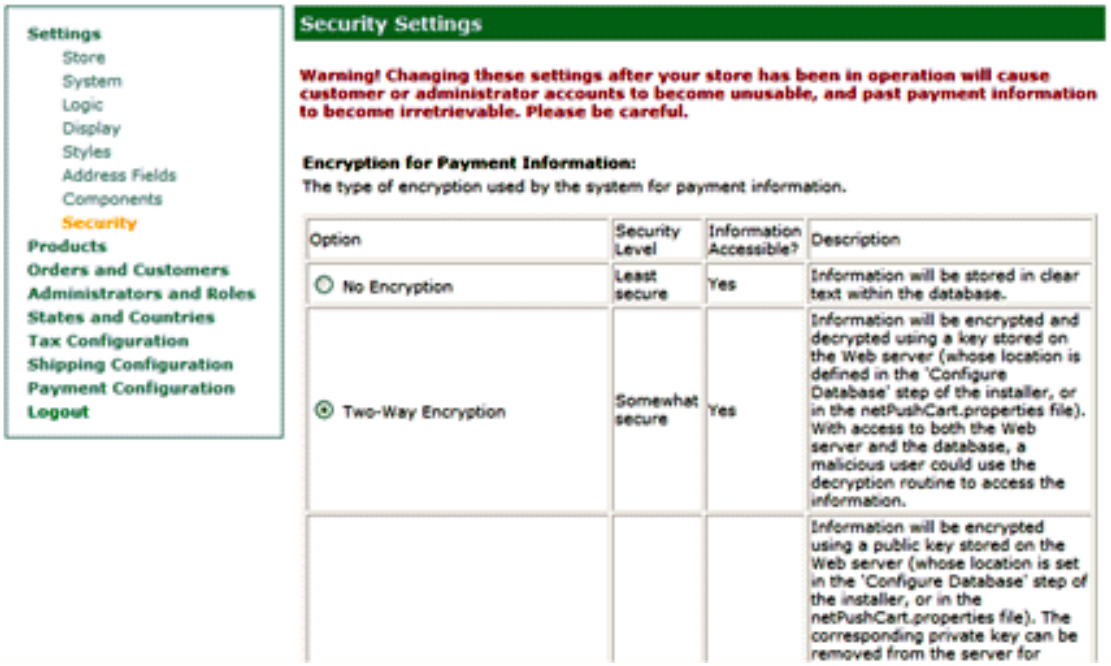
4. If the template still cannot be found, the system uses the `/WEB-INF/layouts/default` directory to find the template.

Tip

For complete details on how to set up multiple custom layouts, refer to the Multiple Custom Layouts section.

Security Settings

The settings defined in the `Settings -> Security` screen should be changed with caution.



Settings -> Security Screen

SoftSlate Commerce does not store the method of encryption at the time a password or payment information is recorded. This has the following consequences, if you decide to switch the encryption settings after your store has been in operation:

- If you have not been using encryption and you switch the settings to use one- or two-way encryption, the system will treat all passwords, including ones defined previously and stored in plain text, as though they are encrypted. If you have to do this, you should first record the unencrytped passwords offline, and then update the passwords using the Administrator, in the `Customers` or `Administrators` area. This will update the customer and adminstrator accounts with the same passwords as before, storing them now in encrypted format.
- Similarly if you have been using two-way encryption for passwords and you switch to no encryption or one-way encryption, none of the previous passwords will be recognized. You should record the passwords offline first, and then update the passwords using the Administrator, in the `Customers` or `Administrators` area so they conform to the new encryption style.
- If you have been using one-way encryption for the passwords, there is no way to recover previous passwords if you then decide to switch the encryption settings. Customers and administrators will not be able to log in after making the switch.
- The same issues arise with respect to credit card information. The system will not be able to decrypt credit card numbers correctly if you change the encryption setting, unless you first record the information offline and then update each of the numbers for each order under the `Orders` area.

Note

The key used for two-way encryption is a file stored on the Web server. The only way to gener-

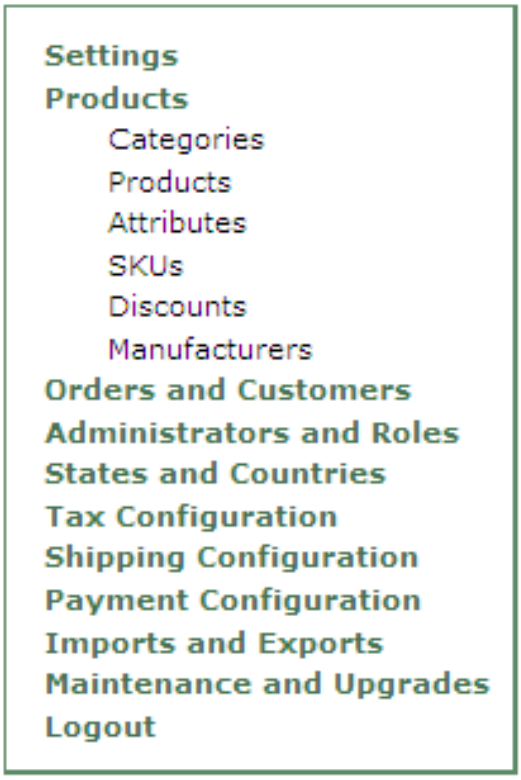
ate this file is through the Installer application. If you need to switch to two-way encryption, but you did not generate the key when you installed SoftSlate Commerce, you need to unlock the Installer tool and run the routine to generate the two-way key.

Chapter 4. Managing the Product Catalog

Overview

The SoftSlate Commerce Administrator application provides the tools you'll use to manage your store's product catalog. The Administrator application is accessed through a browser, by navigating to the URL of your store's installation and appending "/administrator". During installation, you'll create a user name and password to log in to the Administrator.

Upon logging in to the Administrator, click on the "Products" link on the menu to view the various areas you'll use to manage the product catalog:



The Administrator menu

Categories

Use the Categories link to create and modify your store's categories. By creating categories for your store, you'll be able to organize your products, and present a category tree to your customers so they can navigate quickly to the products they're looking for. You can define a parent category for each category you create, which determines its location in the tree. (The categories at the top of the tree are the ones that have no parent.) A parent category can have any number of subcategories, and the tree can go down to an arbitrary number of levels.

Products

Products are the purchasable units of your store, and they are created and modified under the Products link of the Administrator menu. You can assign a "primary category" for each product, which determines which category is shown by default in the breadcrumbing on the product's page. You can also assign products to any number of other categories. On the detail screen of each product, you'll find links to define settings for virtual products, custom settings, inventory settings, and more.

For the detailed steps of creating your first product, and assigning it to a category, visit the Adding Your First Product section.

Attributes

Attributes are customer-selected features that enhance or futher define your products. Use the Attribute link to create and modify them, and to assign them to products. For example, if you sell tee shirts, two typical attributes attached to each tee shirt will be its size and color. Since you want your customers to be able to select the size and color they want to buy, you can set up attributes and options that will appear on the add-to-cart form for them to select. Attributes may appear on the form as radio buttons, a drop down menu, a checkbox, a text field, or a text area. If using radio buttons or a drop down menu, you can also define options to provide choices for your customers. In the tee shirt example, for the size attribute, options of small, medium, large, and extra large would be typical.

For instructions on creating an attribute and options, and assigning it to a product, visit the Making Assignments section.

SKUs

SKUs, or "Stock Keeping Units" are entities you define to work with inventory tracking and discounting. They are created and modified under the SKUs menu. The most common and basic SKU is one that's associated with a product, independent of any its attributes or options. A common practice for many stores is to create one SKU per product in the store. This can be done automatically by running the "SKU Builder" from the SKUs link (selecting the "Build All Product SKUs" option). The current inventory level can then be defined for each product, and discounts can be assigned to each SKU as well.

SKUs also allow you to track inventory and create discounts in a more fine-grained way, because you can also create SKUs that represent a product *and any combination of attributes and options* for the product. For example, if your store sells tee shirts, you'll most likely want to track inventory for large, blue tee shirts separately from the inventory of small, red tee shirts. Similarly, you may wish to create a discount for the extra larges, while continuing to sell the mediums at regular price. By creating SKUs that represent a product plus a selected combination of attributes and options, this level of control can be achieved.

Discounts

Use the Discounts link to create and modify discounts in your store. Discounts of all types are supported, including percentage discounts, absolute value discounts, quantity discounts, and coupons. Administrators have control over numerous settings for each discount, including:

- Which customers the discount applies to
- When the discount starts and if and when it expires
- Which SKU or SKUs the discount applies, or if it applies to the user's cart as a whole.
- How many times the discount may be used

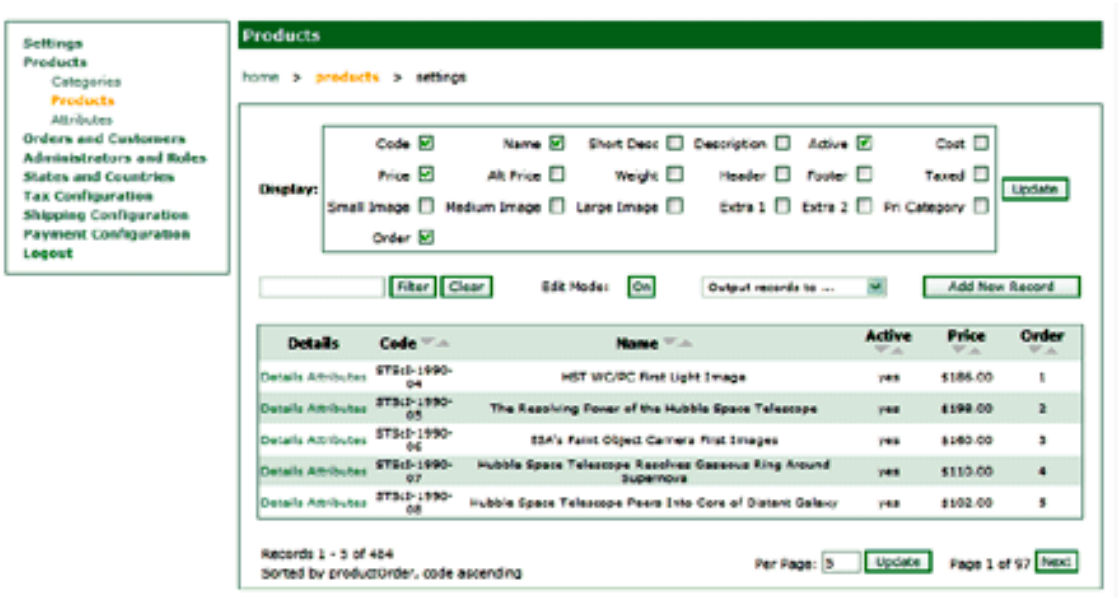
Manufacturers

Manufacturers are companies that make products, or vendors that distribute them, and they are created and modified under the Manufacturers link of the Administrator menu. Once you create one or more manufacturers, you can associate each product with its manufacturer from the product details screen. On the store front, you can display a list of the manufacturers, as well as a detail screen for each manufacturer that lists all of its products. In addition, on each product's detail screen, a link may be placed to allow users to view the manufacturer's details.

Using the Administrator

The Administrator Control Screen

Throughout the Administrator, the same control screen is used when presenting information on products, categories, attributes, orders, customers, and other types of items. The control screen consists of various forms to manipulate what is being displayed and help you maintain the information. A good share of learning how to manage your store through the Administrator is simply learning how to use the features of the control screen.



The Administrator Control Screen for Products

Here is a summary of each of the forms on the control screen:

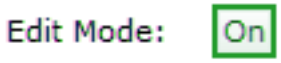
- Display fields form: This feature consists of the big box at the top of the screen. It allows you to control which columns are displayed on the screen. For example, on the `Products -> Products` screen, by default each product's code, name, active status, price, and order are displayed. You can change it so every column of information is presented, or only just a few.

The Display field form

- Filter form: This consists of the box on the left hand side of the screen next to the "Filter" and "Clear" buttons. The form allows you to filter the items that are displayed to just those that match a given search string. For example, to filter products on the `Products -> Products` screen to just those with the word "Jupiter" in the short description, name, or code, type "Jupiter" into the filter form and click the "Filter" button. This can be extremely useful when grappling with a large number of records.

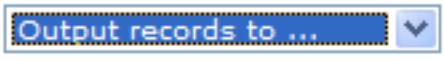
The Filter form

- Edit Mode button: Turning on the Edit Mode button allows you to edit every piece of information that's currently being displayed at once. For example, if you needed to update the prices of all your products at once, you could increase the page size to equal the number of products in the system. (Perhaps you would also want to limit the number of columns being displayed using the Display fields form, to just the Price and Name of the products.) Then click the Edit Mode button. Each of the prices (and any other fields being displayed) are now editable.



The Edit Mode button

- Output records to ... drop down: This feature allows you to create ad hoc reports by outputting the records on the screen, or all the records in the system, to either a printer-friendly format, or to an Excel spreadsheet. For example, to print out a report of the last 50 orders placed in the store, you can go to the `Orders and Customers -> Orders` screen, enter 50 for the page size, sort the items by the order date in descending order, and then select "Print-Friendly Display" in the Output records to ... drop down menu.

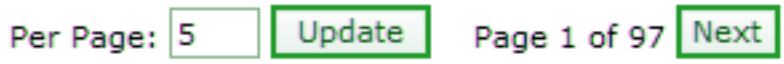


The 'Output Records To ...' menu

- Add New Record button: On each of the screens, you can add a new record to the system--be it a product, order, state, etc.--by clicking the Add New Record button. That will take you to a form where you can add the item, whatever it is.

The Add New Record button

- Sorting arrows: Next to the label of each column, you'll find two grey arrows. Clicking the down arrow sorts the items being displayed in descending order. The up arrow sorts them in ascending order.
- Per Page form/ Next and Previous buttons: The per page form is self-explanatory. It controls the number of items currently being displayed on the screen. Similarly, the Next and Previous buttons help you navigate through the pages.



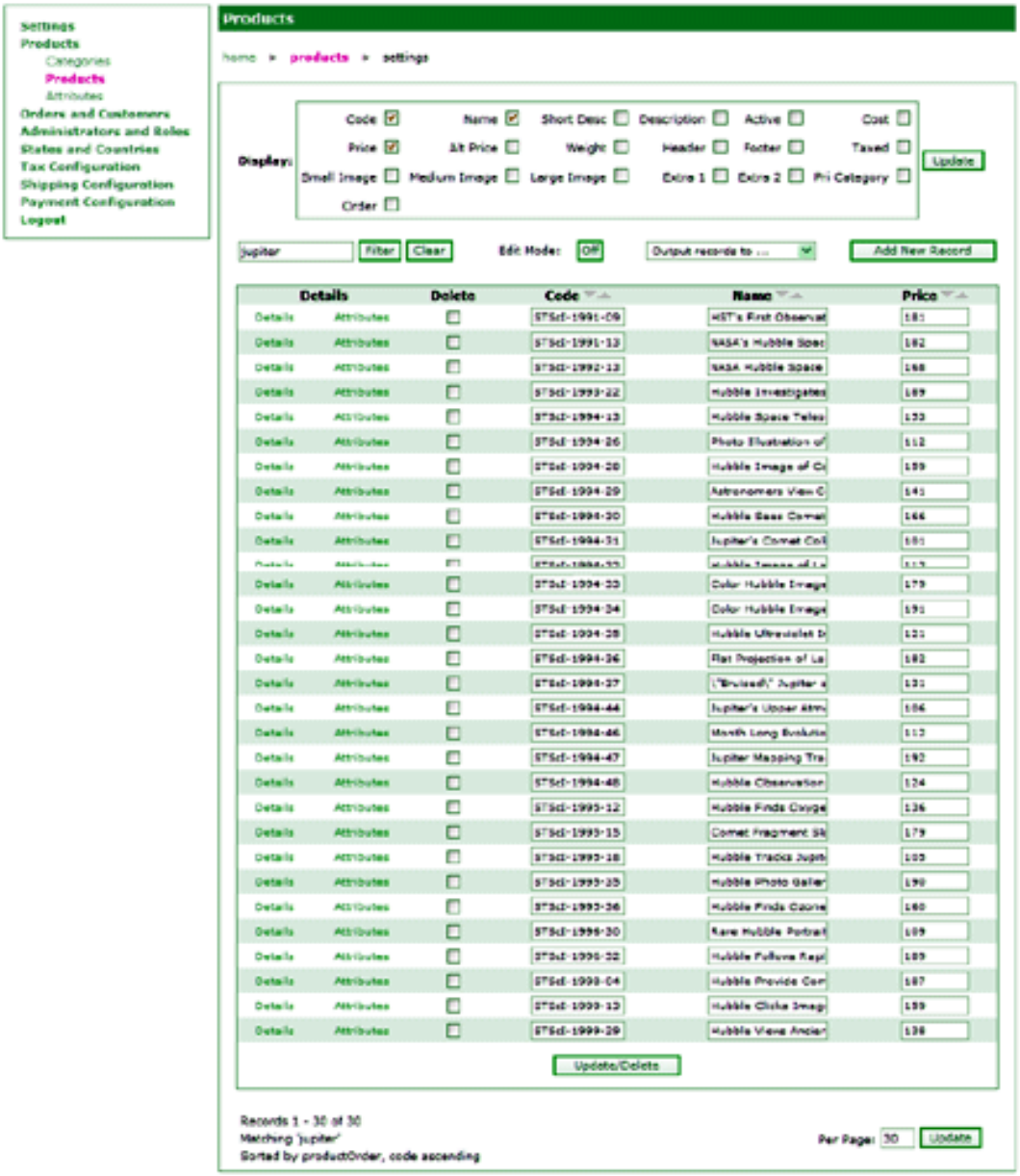
Page Controls

Saving Time with the Administrator Control Screen

One of the ways the control screen features can save a dramatic amount of time is in the case of updating product prices, particularly if your store has a large number of products. The following example shows how to review and update a large group of products all at once. It uses the products from the demo store to demonstrate what to do.

Example 4.1. Updating Prices for a Group of Products at Once

1. Navigate to the `Products -> Products` control screen.
2. For convenience, use the Display form to limit the columns being displayed to just display the Products' code, name, and price. Click the "Update" button next to the box of field names to update.
3. Let's say you want to review all the products containing the word "Jupiter" in the name, code, or short description. Type "Jupiter" into the Filter form, and click the "Filter" button. You should get about 30 results matching the filter.
4. To make things easier, let's update the number of items displayed per page to fit everything on the first page. If you had 30 matches for the filter, change the Per Page form to "30", and click the "Update" button next to it. All the matching products should now appear on one page.
5. Now, click the "On" button in the Edit Mode form to make everything on the screen editable.
6. At this point, the control screen should look like this:



Updating Prices for Multiple Products

As you can see, it's very easy to scan through the list of products and review their prices, changing them when necessary. If you need to, use the sorting arrows next to the column names to organize the products further.

7. When you've finished updating the prices, simply hit the big "Update/Delete" button to have all your changes take effect.

You can use this same technique in the Administrator to make many other sorts of updates or to generate ad hoc reports. For example, it can help you:

- Update the statuses of all the recent orders you've received at once.
- Produce an Excel spreadsheet of all the customers in the store, along with their emails and mailing addresses.
- Produce various reports on the orders you've received, sorted by date, status, billing location, etc.
- Much more.

Making Assignments

In several areas of the Administrator, you need to make "assignments"; for example, assigning products to categories, and assigning attributes to products. In either of these cases, the process is essentially the same. The following steps describe how to create an attribute and assign it to a product. (The process of assigning products to categories is specifically covered in the Adding Your First Product section.)

Example 4.2. Creating an Attribute and Options, and Assigning it to a Product

1. Navigate to the `Products -> Attributes` screen.
2. To add a new attribute, click the "Add New Record Button."
3. An Attribute is used to provide customers with the ability to select options for their products, or enter personalized information. In this example, let's define an attribute named "Color".
4. Use "Color" for the name, and "COLOR" for the code, or whatever else you'd like to use.
5. For the type field, to have the Color options displayed next to each other on the product page using radio buttons, select "Expanded (radio buttons)"; otherwise, you can use "Compact (drop down)" to display the options in the form of a drop-down menu.
6. You can optionally enter in a price, cost, weight, description, etc. for the new attribute.
7. Hit "Add New Record" to add the new attribute to the system.
8. Now navigate back to the `Products -> Attributes` screen. You should see your new attribute in the table in the center of the screen.
9. Click the "Options" button next to the new attribute, and on the susequent screen click the "Add New Record" button to add a new Option under the Color attribute.
10. Add a new option named "Blue". Use "Blue" for the name, and "BLUE" for the code. Optionally, set the option's price, and add a description and image path. The price will be added to the customer's subtotal if he selects this option.
11. Hit "Add New Record" to add the new option to the system.
12. Repeat the above two steps for "Red", "Orange", "Yellow", or whatever other colors you like.
13. Now that you've added the Color attribute, and created the color options under it, you need to assign the attribute to a product. Navigate to `Products -> Products` screen.
14. Click on the "Attributes" link next to the product you wish to add the color attribute to. (If you haven't created any products, click the "Add New Record" button to create one.)

15. You should now see the new Color attribute (along with any other attributes that were already defined) in the table in the center of the screen. To assign one or more of the attributes to the product, click the "On" button next to "Edit Mode".

16. Check the checkbox under the "Assigned" column next to the Color attribute.

17. Click "Update/Delete" to save the change.

Tracking Inventory with SKUs

Overview

You'll find a robust set of features in SoftSlate Commerce related to tracking the stock levels of your products. Store administrators can track inventory for products, and for any combination of products, attributes, and options (SKUs). You can decide to decrement inventory either as users add items to their carts, or as they complete an order. Low stock email notifications can be sent when stock levels reach certain levels. Stock levels are displayed to users, and you can prevent purchases of products that are out of stock, while still displaying them in your store.

To enable inventory for your store, navigate to the `Settings -> Inventory` screen, and set the "Track Inventory" setting to true. You must do this, or none of the inventory features will take effect.

Important

Inventory tracking will not take effect unless the global "Track Inventory" setting on the `Settings -> Inventory` screen is set to true.

The `Settings -> Inventory` screen is also where you define global settings for when to decrement inventory, settings for low stock email notifications, and more.

Note

The "Decrement Behavior" setting defines when the system will subtract quantities from inventory levels. If you choose to decrement as items are added to users' carts, be sure to use the `Maintenance -> Incomplete Orders` screen to delete incomplete orders from the system periodically. When you use that tool to delete the orders, the system will automatically add the order's items back into the inventory levels of the SKUs they were subtracted from.

Most of the settings on this screen can be overridden for an individual product. This is done by clicking the "Inventory Settings" link from the product detail screen (`Products -> Products -> Details`), where you'll find the same set of settings, applicable for just the given product.

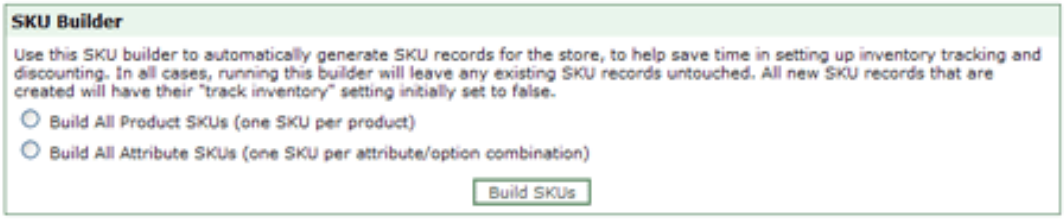
Product-Level Tracking

Now let's go through a basic example of tracking inventory at the product level. For this example, we'll use the demo store, which sells framed images captured by the Hubble space telescope.

Example 4.3. Tracking Inventory at the Product Level

1. The first step for setting up inventory tracking is to turn the global "Track Inventory" flag on from the `Settings -> Inventory` screen, as described above.

2. Assuming we have already created our products, the next step is to create SKU records for each of the products we want to track inventory for. An easy and convenient way to do this is to use the "SKU Builder" at the bottom of the `Products -> SKUs` screen.



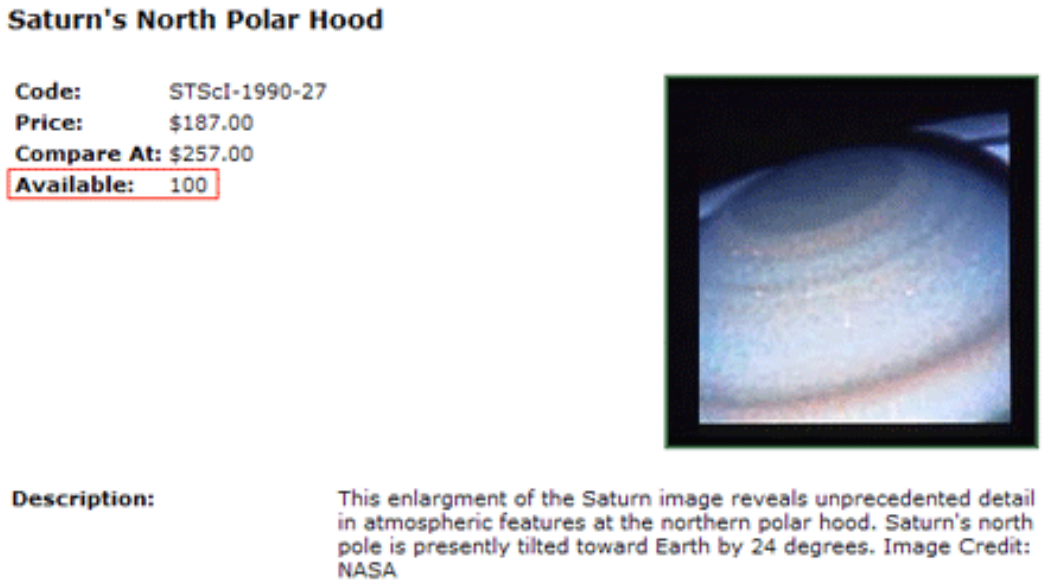
The SKU Builder on the `Products -> SKUs` screen.

3. Using the SKU Builder, create a SKU for each of the products in the store by selecting the "Build All Product SKUs" option. If you have a large number of products, it may take a while so please be patient.

4. The SKU Builder will create a SKU whose "SKU Number" is the same as the product's code. So next, from the `Products -> SKUs` screen find the individual SKU record for the product you want to track inventory on. In our case, we'll use the "Saturn's North Polar Hood" image, whose code is `STScI-1990-27`. You can use the "Filter" form and enter the product's code to help you find it. When you find the record, click on "Details" to view the details for the new SKU.

5. When the SKU builder creates SKU records, the "Track Inventory" flag for the SKU is set to no. This makes it safe to run the SKU Builder without imposing inventory limits right off the bat for every product. From the details screen of the SKU you're adjusting, switch the "Track Inventory" setting to yes, and enter in a positive integer for the "Inventory Level".

6. That's all it takes! In the customer interface for the product, you should now see a new field named "Available" appear below the price, with the current inventory level displayed.



A Product Page Displaying the Product's Current Inventory Level.

A similar field should appear on the product's category page and in search results that match on the product.

Attribute/Option-Level Tracking

While the previous example is a good way to get up and running quickly with basic inventory tracking, in many cases you need to track products differently, if they have attributes that the customer can select. For example, the demo store sells images from the Hubble Space telescope, but also offers different sizes and frames for the images. The "Size" and "Frame" attributes have different options such as small, medium and large, as well as different styles of frame. You might very well need different inventory levels for a large image with a fancy frame as opposed to a small image with a metallic frame. Fortunately, this is possible because you can create different SKUs for each attribute/option combination and track them separately. Let's set this up in our demo store with the following example.

Example 4.4. Tracking Inventory at the Attribute/Option Level

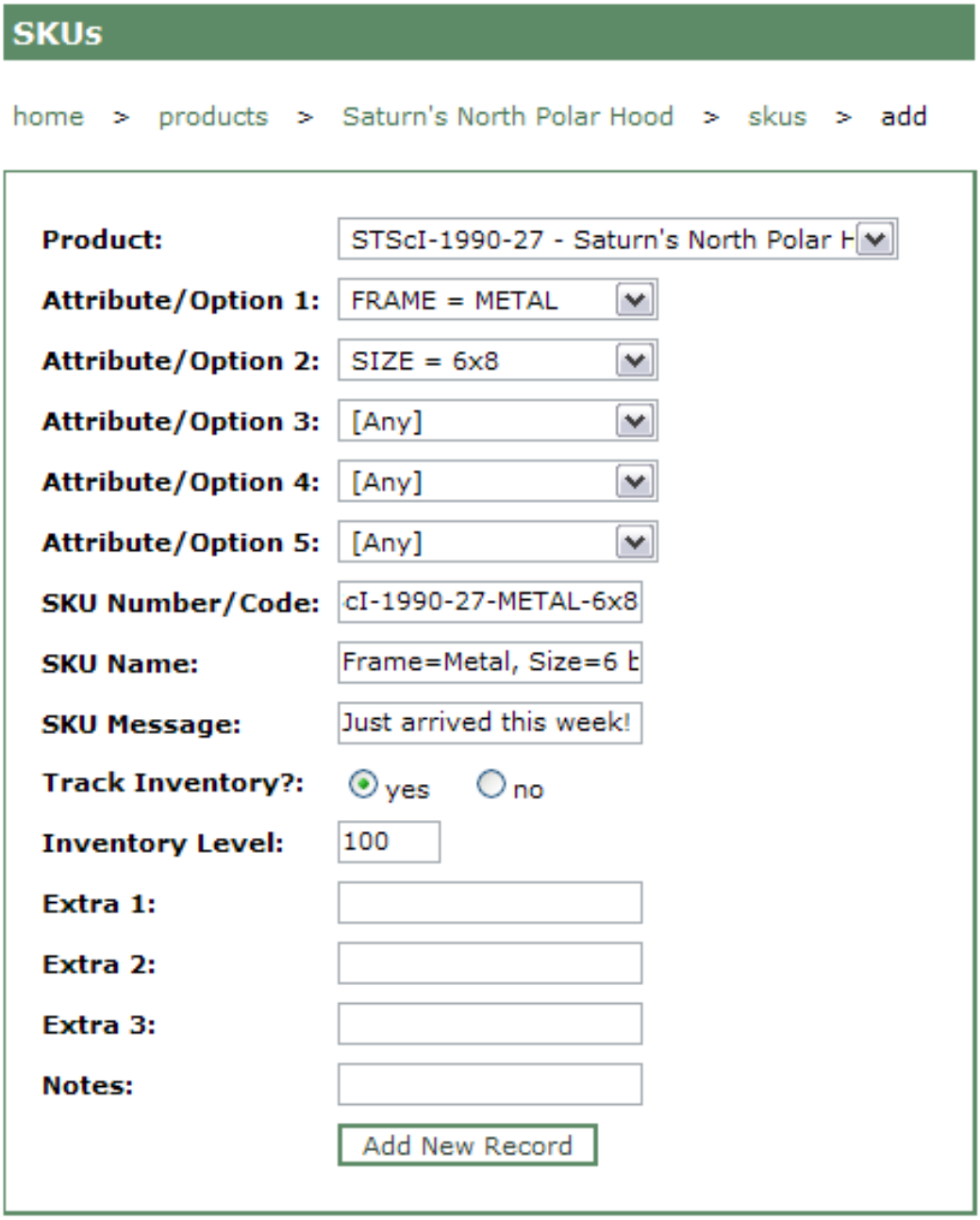
1. To add our new SKU's let's first go the product detail page for the product we're working with. Navigate to the `Products -> Products` screen, and use the "Filter" form to find the product by entering in its code. Click "Details" to go to the product's detail screen. On the top of that screen, find the link that says "SKUs" and click it.

2. The Product SKUs screen shows you all the SKUs related to the given product. You should see the product-level SKU we created in the example above. All we had to do to activate inventory tracking at the product level was to set the "Track Inventory" setting to yes, and enter in an inventory level. But since we now want to track inventory at the attribute and option level, let's switch the SKU's "Track Inventory" setting back to no. You can do this directly from the control screen by turning on "Edit Mode", unchecking the "Track?" checkbox, and clicking the "Update/Delete" button.

3. Now, let's add new SKU records for a specific attribute/option combination for the product. Let's start by defining a SKU for the product whose "Size" is small and whose "Frame" is metal. Click the "Add New Record" button.

4. In the subsequent form, the product we're working with should be pre-selected in the "Product" field. For the "Attribute/Option 1" field, select FRAME = METAL. And for the "Attribute/Option 2" field, select SIZE = 6x8. Then set the "Track Inventory" setting to yes, and enter in a positive number for the inventory level.

5. Before submitting the form, you must define a "SKU Number" and a "SKU Name" for the new SKU. For the "SKU Number" enter a unique identifier for the SKU. In essence, this is the part number for the product plus its attribute/option combination. For the "SKU Name" enter in a descriptive phrase that will tell your customers exactly what the SKU is. In our example, "Frame=Metal, Size=6 by 8" would be a good SKU name. Finally, you can enter in any "SKU Message" to tell your customers anything extra about the status of the SKU's stock. For example, "Just arrived this week!" or "Hurry before we run out of stock". (For out of stock items you could use this field to enter the date you expect the item to come back into stock.) In the end our new SKU will look something like this:

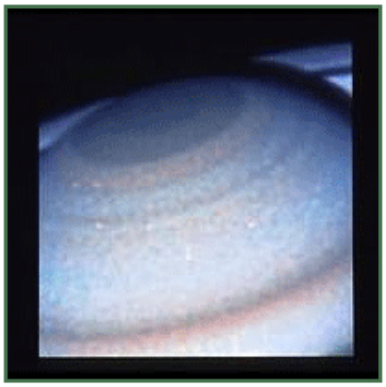


The Add New SKU Form.

6. You can repeat the above steps, altering the values of the "Attibute/Option 1" and "Attribute/Option 2" fields each time, until each of the combinations of size and frame style is represented by its own SKU, with its own inventory level. In the customer interface, you'll be building a display grid that shows your customers the stock levels of all the combinations. The product page will end up looking something like this:

Saturn's North Polar Hood

Code: STScI-1990-27
Price: $187.00
Compare At: $257.00



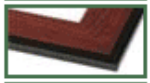
Description: This enlargment of the Saturn image reveals unprecedented detail in atmospheric features at the northern polar hood. Saturn's north pole is presently tilted toward Earth by 24 degrees. Image Credit: NASA

Inventory:

SKU	Available	Message
Frame=Metal, Size=6 by 8	100	Just arrived this week!
Frame=Metal Size=12" by 16"	200	
Frame=Metal Size=24" by 32"	200	
Frame=Wood Size=6" by 8"	200	
Frame=Wood Size=12" by 16"	200	
Frame=Wood Size=24" by 32"	200	
Frame=Fancy Size=6" by 8"	200	
Frame=Fancy Size=12" by 16"	200	
Frame=Fancy Size=24" by 32"	200	

Frame:

(•) Metal +$5.00  A simple, elegant metal frame.

( ) Wood +$6.00  A wood frame for a traditional look.

( ) Fancy +$7.50  A fancy, metallic frame to liven up your study or office.

Size: 6" by 8"

Add To Cart 1

The Product Page with an Inventory Display Grid.

7. That's it, you are now tracking your inventory down to the attribute and option level. Note that by default the inventory grid is not displayed next to each product on the category page or on the search results page - that would make those pages quite crowded. However, if there is only one attribute/option defined for a SKU, the available inventory level will appear next to the option on all

drop down menus and radio buttons.

Attribute-Only SKUs

There is one more important variation of our previous examples. In some cases, your store might stock a given attribute independently of any products it's assigned to. In fact, the "Frame" and "Size" attributes of the demo store are good examples. If we ran our store in such a way that an image was printed as each order arrived, and then matched with a large metal frame, or a small fancy frame, etc., we might very well want to track the frames completely separately from any of the products. In other words, if we have 1,000 large metal frames, that's the number we have for the entire store, not just for a given product.

Fortunately, it is easy to set up attribute-only SKUs: SKUs that apply to a combination of attributes and options, independent of any products. The easiest way to do so is to go back to the SKU Builder on the `Products -> SKUs` screen. This time, select the "Build All Attribute SKUs" option. The SKU Builder will create a new SKU record for each attribute and each of its options. You can then turn on the "Track Inventory" setting for the SKU and assign it an inventory level. The store will display the attribute's inventory levels in the SKU grid the same way it displays other SKUs, on the product page of products that have the attribute assigned to it.

Discounts

Overview

SoftSlate Commerce supports a variety of discount techniques that you can apply to the products in your store. The types of discounts it supports include percentage discounts, absolute value discounts, quantity discounts, and coupons. You'll also find a number of settings allowing you to control who gets each discount, when the discount starts and expires, what products it applies to, and how many times each discount may be used.

To enable discounting for your store, navigate to the `Settings -> Discounts` screen, and set the "Apply Discounts" setting to true. You must do this, or none of the discount features will take effect.

Important

Discounts will not take effect unless the global "Apply Discounts" setting on the `Settings -> Discounts` screen is set to true.

You'll also find a few other settings on the `Settings -> Discounts` screen that control the overall behavior of discounts.

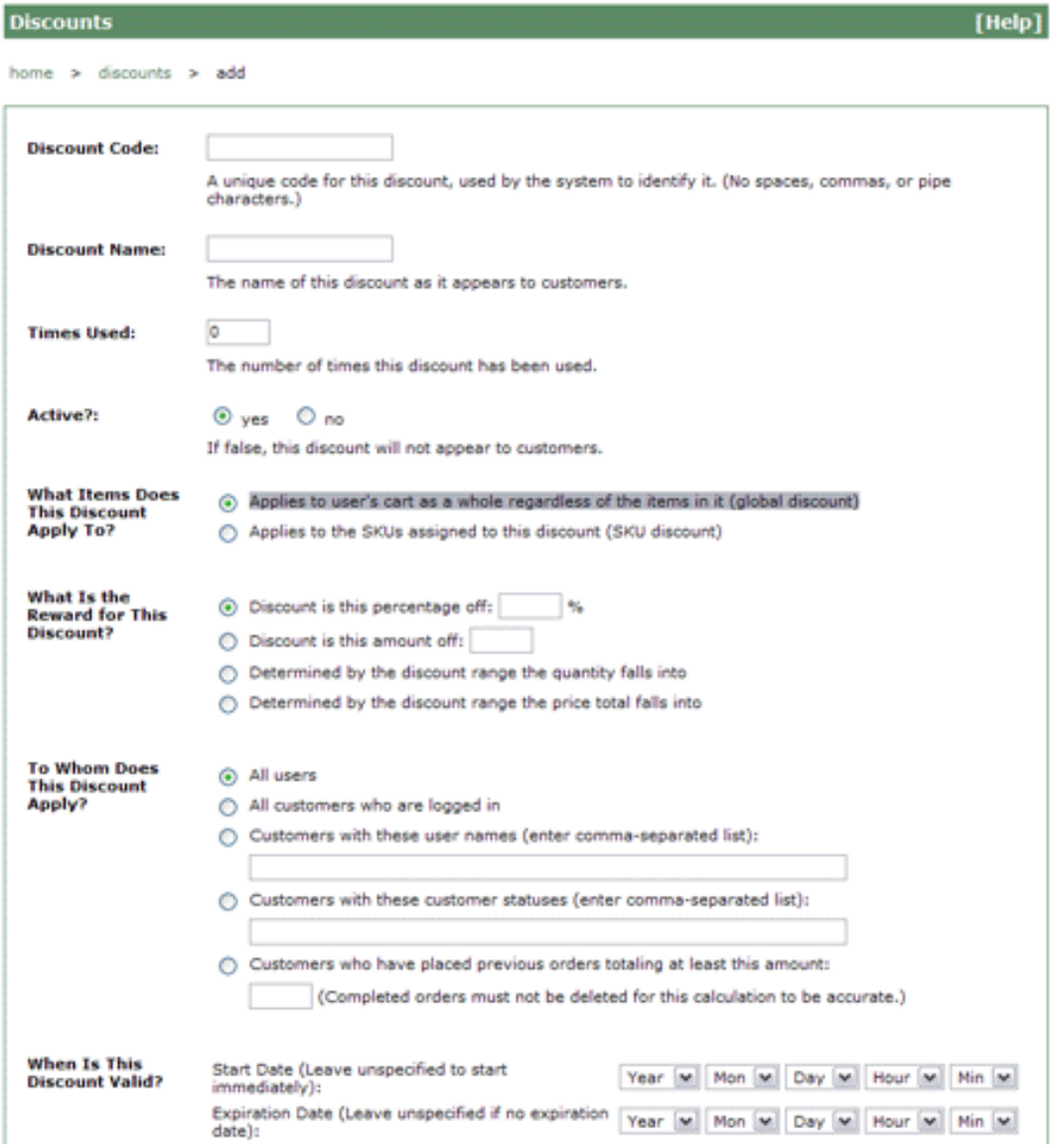
Global Discounts

A global discount is defined as one that does not apply to a restricted set of products or SKUs, but instead applies to the customer's cart as a whole. A simple example would be a sale in which customers receive 10% off everything in the store. A global discount of 10% would be created, that applies to the customer's cart as a whole, rather than to individual items. Let's go through the steps involved setting up this sort of discount in the following example.

Example 4.5. Setting up a Global 10% Discount

1. The first step for setting up any discount is to turn the global "Apply Discounts" flag on from the `Settings -> Discounts` screen, as described above.

2. Navigate to the add discount form by going to the `Products -> Discounts` screen and clicking the "Add New Record" button.



The Add Discount Form.

3. The add discount form should be fairly self-explanatory. To create our global 10% discount, first enter a unique discount code and give the discount a name.

4. For the "What Items Does This Discount Apply To?" field, select the "Applies to user's cart as a whole regardless of the items in it (global discount)" option to identify this as a global discount.

5. For the "What Is the Reward for This Discount?" field, select the "Discount is this percentage off:" option, and enter "10" in the percentage field.

6. For this example and in many cases generally, you can leave the other fields as they are. But note

that these other options are available to control the specific customers that can use the discount, the start time and expiration time of the discount, and how many times it can be used.

7. That is all it takes! If you go to the customer interface for the store, as soon as you add a product to the cart, you'll see that the global discount appears as a separate line item in the cart. The discount's total is a negative amount equal to 10% of the cart's subtotal.

Note

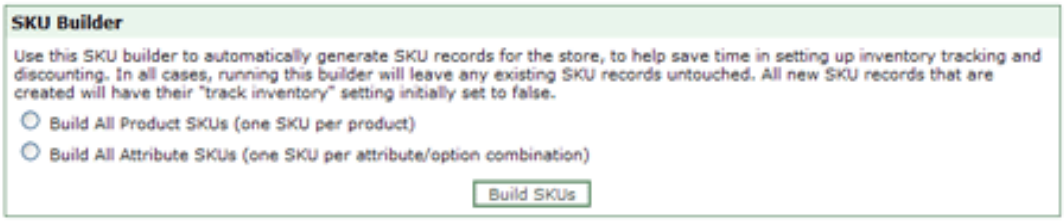
Global discounts are applied against the order's *subtotal*. They will not take shipping or taxes into account.

SKU Discounts

Unlike global discounts, SKU discounts apply to one or more specific SKUs. As we learned in the section on Tracking Inventory with SKUs, a SKU (or "Stock Keeping Unit") can represent a product, or if needed, any combination of attributes and options for a product. In the following example, let's set up a discount for a basic product-level SKU.

Example 4.6. Setting up a SKU Discount to Give $10.00 off

1. Assuming we have already created some products, the first step is to create SKUs that represent them. An easy and convenient way to do this is to use the "SKU Builder" at the bottom of the `Products -> SKUs` screen.



   The SKU Builder on the `Products -> SKUs` screen.

2. Using the SKU Builder, create a SKU for each of the products in the store by selecting the "Build All Product SKUs" option. If you have a large number of products, it may take a while so please be patient.

3. Next, let's make sure that the global "Apply Discounts" flag setting is on by checking the `Settings -> Discounts` screen.

4. Now navigate to the add discount form by going to the `Products -> Discounts` screen and clicking the "Add New Record" button.

5. For the "What Items Does This Discount Apply To?" field, select the "Applies to the SKUs assigned to this discount (SKU discount)" option to identify this as a SKU discount.

6. For the "What Is the Reward for This Discount?" field, select the "Discount is this amount off:" option, and enter "10" in the text field.

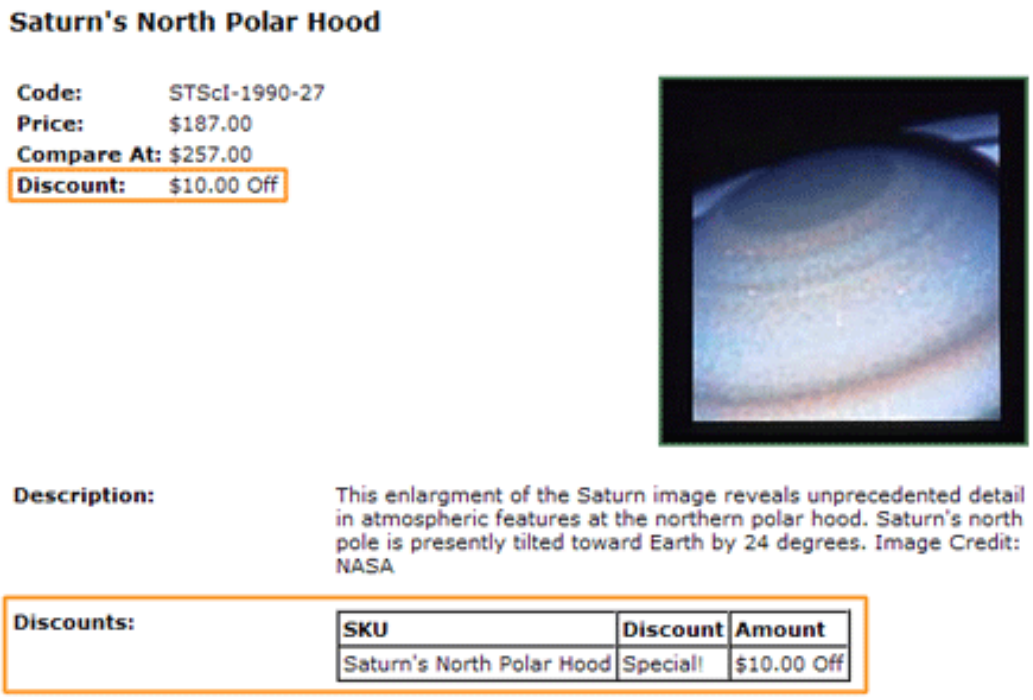
7. For this example and in many cases generally, you can leave the other fields as they are. But note that these other options are available to control the specific customers that can use the discount, the start time and expiration time of the discount, and how many times it can be used. For now, let's go ahead and add the discount.

8. Now that our SKUs have been created, and our discount is in place, the last thing to do is to assign the SKUs we want to the discount. From the `Products -> Discounts` screen, click the "Details" link next to our new discount.

9. On the edit discount screen, click the "SKUs" link. This should take you to the SKU Discount assignment screen. You'll see a filterable list of the available SKUs that we created.

10. To assign one or more of the SKUs to our discount, simply switch the "Edit Mode" to on, check off the SKUs you'd like to assign, and click the "Update/Delete" button.

11. That's all it takes! On the product page in the customer interface, you should now see a new field named "Discount" appear below the price, with the current discount displayed. In addition, a discount grid will appear that lists all the discounts applicable for the product.



A Product Page Displaying the Applicable Discounts.

Quantity Discounts and Discount Ranges

A quantity discount is one which has different Discount Ranges based on the quantity (or price total) of items being purchased. For example, a deal where the customer gets 5% off a product if he orders 10 or more of that product, and 10% off if he orders 20 or more, is a simple example of a quantity discount. Building on our last example, let's go through the steps involved setting up this sort of discount.

Example 4.7. Setting up a Quantity Discount

1. Assuming we have set up a SKU discount like the one in the above example, the first step is to re-define our discount as one which looks at its Discount Ranges to determine the amount of the reward. Navigate to the edit discount form by going to the `Products -> Discounts` screen and clicking the "Details" link next to our SKU discount.

2. Change the "What Is the Reward for This Discount?" field for our discount to "Determined by the discount range the quantity falls into". This tells the discount to look at the Discount Ranges under it to figure out the amount of the discount.

3. Now we must create our Discount Ranges under the discount. From the edit discount form, click on the link that says "Discount Ranges".

4. On the subsequent screen, click the "Add New Record" button to go to the add Discount Range form.

5. For the "Floor for This Range" field, enter 5. Our first range will take effect only if the customer is purchasing at least 5 of the item.

6. For the "Ceiling for This Range" field, enter 9. This marks the upper limit of the first range.

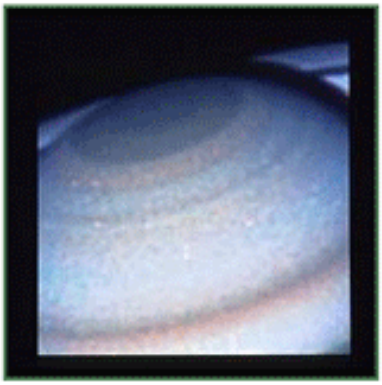
7. For the "Reward for This Range" field, select the "This percentage off" option, and enter 10 for the percentage. Click Add to add the new Discount Range.

8. Repeat the above steps to add the parameters of our second range. Enter 10 for the floor of the range, and leave the ceiling at 0, which is used to represent no ceiling. For the reward, enter 20 for the percentage.

9. That's it! In the customer interface, note that on the product page, the discount grid now included the two ranges and explains the discounts that the customers will receive depending on how many of the item they buy.

Saturn's North Polar Hood

Code: STScI-1990-27
Price: $187.00
Compare At: $257.00



Description: This enlargment of the Saturn image reveals unprecedented detail in atmospheric features at the northern polar hood. Saturn's north pole is presently tilted toward Earth by 24 degrees. Image Credit: NASA

Discounts:

SKU	Discount	Amount
Saturn's North Polar Hood	Special!	Quantities 5 to 9 : 10.0% Off 10 and up : 20.0% Off

A Product Page Displaying a Quantity Discount.

Note

You can make a quantity discount a global discount. In that case, the quantity used to find the applicable Discount Range is the total number of items the customer has in the cart.

Coupons

In SoftSlate Commerce, any discount can be easily turned into a coupon. This is done by entering a unique code into the "What Is This Discount's Coupon Code?" field on the discount add/edit form.

Defining a coupon code for the discount tells the system to apply the discount only if the customer enters the code into the coupon form in the customer interface. The coupon form can appear either on the view cart screen, or during checkout above the address form. Navigate to the `Settings -> Discounts` screen to define which location, or both, should display the coupon form.

Virtual Products

With 2.x, SoftSlate Commerce supports selling "virtual products," or products that are downloadable as files after purchase. Examples include a music store selling mp3 files, or a software vendor selling downloadable software programs.

After a customer places an order for a virtual product, he can log in and download any number of files associated with the products he's bought. The files are only available to customers who have created an account, after logging in. For this reason, you should consider setting the "Force Login on Orders" setting to true on the Settings -> Logic screen if you're selling virtual products.

Here is a summary of how to set up virtual products:

1. In the Administrator, create a product as you would any other product, by navigating to `Products -> Products` -> Add New Record.
2. Go into the details screen of the new product (`Products -> Products` -> Details). You'll see one of the links across the top is for "File Downloads". Click that to create a file associated with the product.
3. The fields you're filling in are:
   - File Name: The name that customers will see for the file when they download it (which can be different than the actual name of the file on the server).
   - Full Path on Server: The full path to the file as it lives on the web server. To ensure that the file is only accessible to those who have bought it, you should consider placing it outside the web root.
   - Mime Type: The mime type that the file will be presented with to the user's browser. (For binary files, you might try "application/octet-stream".)
   - Description: Displayed to the user on the download screen
4. Now as a customer you can place an order for the product. After placing the order, browse to "Ac-

countDownload.do" for the download screen, where you should see a link to download all the files associated with the products in the order.

5. You'll probably want to add a link to the download screen to the customer account menu. Create a custom version of the `/WEB-INF/layouts/default/customer/accountMenu.jsp` file to display the link to AccountDownload.do. (By default the link is not displayed because most stores don't have virtual products.) Copy this file into the `/WEB-INF/layouts/custom/customer` directory and add the link to the menu and it should appear in the customer's account area, next to the links to edit the account settings.

Advanced Search With Lucene

With 2.2.5, SoftSlate Commerce supports a more advanced method of searches against your product catalog by your customers. Lucene [http://lucene.apache.org/] is an open-source technology from the Apache Software Foundation. By default, the search form customers use hits the database directly with an SQL query to pull in the products that match the search string. While that is suitable for many small sites, using Lucene provides a more sophisticated search that pulls in more relevant results. Plus, with Advanced Search you can sort the search results by relevance and price.

To set up Advanced Search with Lucene in SoftSlate Commerce, simply follow these steps:

1. The first step is to create a search index for Lucene to use. The search index takes all the product information in the database and places it in special files Lucene uses to look up the search results. To create the index, in the Administrator navigate to `Products -> Advanced Search` and click the "Rebuild Index" button.

Important

The search index captures your product information only at the time you click the "Rebuild Index" button. If you subsequently add or update products, you must rebuild the index again to have the new information appear in search results.

By default, SoftSlate Commerce places the index files inside the `/WEB-INF/lucene` directory. You can nominate a different location however, by adjusting the "Lucene Index Directory" setting.

2. Next, in the Administrator, navigate to the global Advanced Search settings screen at `Settings -> Advanced Search`. Here you can define which areas of the application will use the Advanced Search. In particular, you can use Lucene for searches customers enter into the search box, for the display of category pages (allowing for sorting products by price), and other areas. Turn on the areas you which to use Advanced Search for.

3. That's it. Your store now has the Advanced Search feature enabled. Take it for a test drive, and don't forget to rebuild the index whenever you add or update product information.

Chapter 5. Configuring Taxes

SoftSlate Commerce is distributed with the ability to set up a tax table based on the location of each customer's delivery address. You can define separate tax rates by country, state, city, and postal code.

Basic Tax Rates

For the following example, let's suppose we operate a store in Boulder, Colorado. Our local laws tell us that we must charge tax for any sales we make to customers located in Colorado, and additional taxes on top of that for customers in Boulder County, and another tax on top of that for customers in the city of Boulder.

Example 5.1. Configuring Taxes for a Store in Boulder, Colorado

1. Navigate to the `Tax Configuration -> General Settings` screen.
2. Make sure that the "Are Tax Rates Cumulative" setting is set to yes. This tells the store to apply all the tax rates that match the customer's delivery address.
3. Make sure the "Basic Tax Rates" checkbox is checked.
4. Click "Save" to update the settings.
5. Now navigate to the `Tax Configuration -> Basic Tax Rates` screen.
6. Click the "Add New Record Button" to begin adding tax rates for your store.
7. Add the first tax rate for all customers in Colorado. For "Country", select "United States" and for "State" select "Colorado". For "Amount", enter "2.9" for 2.9% state sales tax. Leave the other fields as "[Any]".

   #### Note

   The actual tax rates may be different than those used in this example. Please research the current tax rates for your area.

   Click the "Add New Record" button to add the first rate. This first rate will apply to all orders whose delivery addresses are in the United States and Colorado.
8. Now begin adding the tax rates for customers in Boulder County. To do this, we must find out all the postal codes that correspond to Boulder County. Research tells us the following postal codes apply: 80510, 80329, 80328, 80323, 80322, 80321, 80314, 80310, 80309, 80308, 80307, 80306, 80305, 80304, 80303, 80302, 80301, 80038, 80021, 80020, 80025, 80516, 80533, 80455, 80026, 80504, 80503, 80502, 80501, 80028, 80027, 80540, 80466, 80544, 80471, 80481.

   Add a tax rate record for each of the above postal codes. For each one, select "United States" as the country and "Colorado" as the state. For "Amount", enter ".65" for .65% county sales tax. Leave the city field marked as "[Any]".

   Click the "Add New Record" button to add each of the rates for Boulder County.
9. Finally, add a tax rate for the city of Boulder sales tax. For "Country", select "United States", for

"State" select "Colorado", and for "City" type in "Boulder". For "Amount", enter "3.41" for 3.41% sales tax. Leave the postal code field as "[Any]".

Click the "Add New Record" button to add the tax rate for the city of Boulder.

10. With these rates set up, each time a customer comes through checkout and uses a delivery address in Colorado, the tax rate corresponding to Colorado will be applied. If the delivery address's postal code matches one of Boulder County's postal codes, the county tax rate will also be applied. And if the delivery address's city matches "Boulder", the city tax rate will also be applied.

Note

Postal codes do not always line up precisely with county boundaries, and sometimes an address might use a given city for its mailing address but actually be outside that city's tax district. In other words, using Basic Tax Rates may not be 100% accurate for 100% of your customers, but it is likely to be accurate for the vast majority.

Chapter 6. Configuring Shipping

SoftSlate Commerce is distributed with the ability to set up several different kinds of shipping tables to determing the shipping costs for your customers. You can define shipping tables based on the quantity, total price, or total weight of the customer's order. In addition, you can define a flat rate for shipping that applies regardless of the quantity, weight, or price.

Flat Rate Shipping Methods

Let's suppose you operate a store that has very simple rules for shipping costs: if the customer wants their order to be rushed to them, you charge a flat rate of $9.95 for shipping. Otherwise, you charge $5.95. Setting up this type of shipping is quite simple.

Example 6.1. Charging Flat Rates for Shipping

1. Navigate to the `Shipping Configuration -> Shipping Methods` screen.
2. Click "Add New Record" to add the first shipping method, for standard, 5.95 shipping.
3. Use "Standard" for the name of the method, "STD" for the code, leave Minimum Charge at 0, but change Base Charge to 5.95.

Note

Leave dollar signs or other currency symbols off when entering price values.

4. Under Method Type, select "Flat Rate".
5. Click "Add New Record" to add the new method to the store.
6. Now add the next shipping method, for rushed, 9.95 shipping.
7. Use "Rushed" for the name of the method, "RSH" for the code, leave Minimum Charge at 0, but change Base Charge to 9.95.
8. Under Method Type, again select "Flat Rate".
9. Click "Add New Record" to add the new method to the store.
10. Navigate back to the `Shipping Configuration -> Shipping Methods` screen.
11. You should see both of your methods listed in the table at the center of the screen. Customers will see both options during checkout and can select either Standard (5.95) shipping, or Rushed (9.95) shipping for their order.

Important

No shipping methods will show up during checkout to users unless you have switched the "Is a Shipping Selection Required?" setting from no to yes on the `Settings -> Logic` screen.

For more details, refer to the section on the Shipping Selection Required Setting.

Quantity-Based Shipping Tables

It's common for stores to base the amount they charge for shipping on the quantity of items the customer is ordering. For the following example, let's suppose you operate a store where you would like to charge a base of $5.00 handling for every order. In addition to that, you'd like to charge $1.50 for each of the first three items to be shipped, plus $0.50 for every item after the third. This sort of policy corresponds to a "Cumulative Quantity Table" shipping method type.

Example 6.2. Charging Shipping Based on the Quantity of Items in the Order

1. Navigate to the `Shipping Configuration -> Shipping Methods` screen.
2. Click "Add New Record" to add the shipping method.
3. Use "Standard" for the name of the method, "STD" for the code.
4. Leave Minimum Charge at 0, but change Base Charge to 5.00. This corresponds to the amount of handling charges you want applied to every order.

   ## Note

   Leave dollar signs or other currency symbols off when entering price values.

5. Under Method Type, select "Cumulative Quantity Table".
6. Click "Add New Record" to add the new method to the store.
7. Navigate back to the `Shipping Configuration -> Shipping Methods` screen.
8. You should see your new method listed in the table at the center of the screen.
9. Click the "Rates" link next to the new method to add the two levels of quantities you need.
10. Click "Add New Record" from the "Rates" screen to add the first quantity range.
11. Use 0 for the floor of the first range, 3 as the ceiling, and use 1.50 for the amount of this first quantity range.

    This tells the store to use this rate if the number of items in the user's cart is greater or equal to 0 and less than or equal to 3.
12. Click "Add New Record" to add the first rate to the shipping method.
13. Now enter the next range in the quantity-based shipping table. Use 4 for the floor of the first range, 0 as the ceiling, and 0.50 as the amount.

    This tells the store to use this rate if the number of items in the user's cart is greater or equal to 4, up to infinity (0 as the ceiling means no upper limit).

    ## Important

Ranges for the Cumulative Quantity Table must not overlap. In other words, make sure the floor for one range is larger than the ceiling for the previous range. Otherwise, the amounts from both ranges will be applied when computing a customer's shipping cost.

14. Click "Add New Record" to add the second rate to the shipping method.

As an example of how this works, if a user has 5 items in his or her cart, the store will compute the shipping costs for this method to be 5.00 (from the Base Charge for the method), plus 1.50 for the first item, plus 1.50 for the second item, and plus another 1.50 for the third. At this point the second range will kick in, and the store will add 0.50 for the fourth item and another 0.50 for the fifth. The total shipping cost presented to the customer is 10.50.

Sometimes it's simpler and more convenient to set a static rate for each quantity range, rather than having the costs accumluate for each item as they do in this example. This is done by selecting the regular "Quantity Table" for the Method Type of the shipping method instead of "Cumulative Quantity Table". In that case, the number you enter for the "Amount" of each quantity range is the *flat amount* that will be charged if the user's order falls within that range (plus the Base Charge), rather than the *price per item*.

Important

No shipping methods will show up during checkout to users unless you have switched the "Is a Shipping Selection Required?" setting from no to yes on the `Settings -> Logic` screen. For more details, refer to the section on the Shipping Selection Required Setting.

Weight-Based Shipping Tables

Another common practice is for stores to base the amount they charge for shipping on the total weight of items the customer is ordering. For the following example, let's suppose you operate a store where you would like to charge a base of $5.00 handling for every order. In addition to that, you'd like to charge $1.00 per pound for orders whose weights total between 0 and 10 pounds. For heavier orders totaling more than 10 pounds, you'd like to charge $0.75 per pound. This sort of policy corresponds to a "Per Weight Unit Table" shipping method type.

Example 6.3. Charging Shipping Based on the Weight of Items in the Order

1. Navigate to the `Shipping Configuration -> Shipping Methods` screen.
2. Click "Add New Record" to add the shipping method.
3. Use "Standard" for the name of the method, "STD" for the code.
4. Leave Minimum Charge at 0, but change Base Charge to 5.00. This corresponds to the amount of handling charges you want applied to every order.

Note

Leave dollar signs or other currency symbols off when entering price values.

5. Under Method Type, select "Per Weight Unit Table".

6. Click "Add New Record" to add the new method to the store.
7. Navigate back to the `Shipping Configuration -> Shipping Methods` screen.
8. You should see your new method listed in the table at the center of the screen.
9. Click the "Rates" link next to the new method to add the two levels of weights you need.
10. Click "Add New Record" from the "Rates" screen to add the first weight range.
11. Use 0 for the floor of the first range, 10 as the ceiling, and use 1.00 for the amount of this first weight range.

    This tells the store to use this rate if the total weight of items in the user's cart is greater or equal to 0 and less than or equal to 10.
12. Click "Add New Record" to add the first rate to the shipping method.
13. Now enter the next range in the weight-based shipping table. Use 10 for the floor of the first range, 0 as the ceiling, and 0.75 as the amount.

    This tells the store to use this rate if the total weight of items in the user's cart is greater or equal to 10, up to infinity (0 as the ceiling means no upper limit).

Important

Ranges for the Per Weight Unit Table can overlap, in which case the rate of the lower range will be used. In other words, it's ok to make the ceiling for the lower range equal to the floor for the following range. Because weights are computed as decimal values, doing this will ensure that no order will fall through the cracks (i.e. it will make sure all possibilities are accounted for if the order's weight is something like 10.01).

14. Click "Add New Record" to add the second rate to the shipping method.

As an example of how this works, if a user is placing an order and the total weight of all the items in the order is 15.25 pounds, the store will compute the shipping costs for this method to be 5.00 (from the Base Charge for the method), plus 15.25 times 0.75 (because 15.25 falls in the second range). The shipping cost presented to the customer will be 16.44.

Sometimes it's simpler and more convenient to set a static rate for each weight range, rather than having the costs be multiplied out per unit of weight as they are in this example. This is done by selecting the regular "Weight Table" for the Method Type of the shipping method instead of "Per Unit Weight Table". In that case, the number you enter for the "Amount" of each weight range is the *flat amount* that will be charged if the user's order falls within that range (plus the Base Charge), rather than the *price per unit of weight*.

Important

No shipping methods will show up during checkout to users unless you have switched the "Is a Shipping Selection Required?" setting from no to yes on the `Settings -> Logic` screen. For more details, refer to the section on the Shipping Selection Required Setting.

Price-Based Shipping Tables

Many stores find they need to correspond the shipping costs they charge to the total price of the customer's order. For example, let's suppose you would like to reward customers who make large purchases in your store by offering free shipping on orders of $200 or more. You can set this up quite easily by creating a shipping method with a "Price Table" method type.

Example 6.4. Offering Free Shipping on Large Orders

1. Navigate to the `Shipping Configuration -> Shipping Methods` screen.
2. Click "Add New Record" to add the shipping method.
3. Use "Free Shipping for Orders Over $200" for the name of the method, "FREE" for the code.
4. Leave Minimum Charge at 0, and leave the Base Charge at 0.

Note

Leave dollar signs or other currency symbols off when entering price values.

5. Under Method Type, select "Price Table".
6. Click "Add New Record" to add the new method to the store.
7. Navigate back to the `Shipping Configuration -> Shipping Methods` screen.
8. You should see your new method listed in the table at the center of the screen.
9. Click the "Rates" link next to the new method to add your free shipping rate to the method.
10. Click "Add New Record" from the "Rates" screen to add a range corresponding to the free shipping offer.
11. Use 200 for the floor of the range, and 0 as the ceiling, and use 0 for the amount of this range.

    This tells the store to use this rate if the total price of items in the user's cart is greater or equal to 200, up to infinity (0 as the ceiling means no upper limit).
12. Click "Add New Record" to add the rate to the shipping method.

Now when a customer goes through checkout the store will check to see if the total price of items in the order is greater or equal to 200. If it is, it will display this method, "Free Shipping for Orders Over $200", as one of the options for the customer to choose.

Important

No shipping methods will show up during checkout to users unless you have switched the "Is a Shipping Selection Required?" setting from no to yes on the `Settings -> Logic` screen. For more details, refer to the section on the Shipping Selection Required Setting.

Shipping Rules

Overview

The Shipping Rules feature is accessible from the `Shippping Configuration -> Shipping Rules` screen of the Administrator. The purpose of shipping rules is to allow you to create a number of business rules that can be applied to any of the shipping methods active in the store, for specific SKUs, and for specific customer delivery locations. Specifically, there are four flavors of shipping rules, which we will cover in the following sections.

Package Shipping Rules

Perhaps the most powerful type of shipping rule is the "Package" shipping rule. The purpose of Package rules is to generate more accurate shipping quotes from the built-in integrations with UPS and USPS. The UPS and USPS processors have the ability to generate shipping rates for shipments that have multiple packages. By creating a set of Package shipping rules, you can define how your products are divided into packages when you ship them. These package definitions can then be made part of the rate request and provide more accurate rates for your customers. The Administrator interface allows you to define the quantities of different products that go into a package, and what the weight and dimensions of each package is. Let's illustrate this with an example.

Example 6.5. Defining Packages with Package Shipping Rules

In this example, let's say we run a store that sells coffee mugs. While many of our customers purchase just one mug when they buy from us, some purchase many more than one, for example, if they need to buy mugs for all the workers in their office. When we stock our mugs, we know that they come to us in boxes of 60 mugs each, and within those big boxes, there are smaller packages of 6 mugs each. If we get an order for 60 mugs, we'll use the big box for the package. If we get an order for 6 mugs, we'll use one of the smaller boxes. For anything that's not a multiple of 6 or 60, we'll ship the remainder in our own packaging. With this knowledge, we can create a set of Package shipping rules to help us provide the most accurate UPS and USPS quotes possible for customers during checkout.

1. To create a Package shipping rule, navigate to the `Shippping Configuration -> Shipping Rules` screen, and click the "Add New Record" button. On the subsequent form, select "Package Size or Weight Rule" for the "Type of Shipping Rule" field. Give your shipping rule a name to help you keep track of it and click the "Add New Record" button to save it.

2. Now navigate back to the `Shippping Configuration -> Shipping Rules` screen and click on the "Details" link next to the new rule you created.

3. First, we'll define which products the new rule applies to. From the shipping rule details screen, click the "SKUs" link.

4. Package shipping rules are associated with SKUs as opposed to products themselves. This allows you to define different rules for different attribute and option combinations for the same product. For this example however, let's assume there is just one coffee mug SKU that we're creating the shipping rule for. If you have not yet created SKU records for the coffee mug products, you would do so now by running the "SKU Builder" at the bottom of the SKU control screen by selecting the "Build All Product SKUs" option.

5. With the SKU created, assign it now to the shipping rule by clicking the "Edit Mode" to "on" on the SKU control screen, and checking off the "Assigned" checkbox for the SKU.

6. We've associated the shipping rule with the product. Next, we need to define the package details for the rule. Navigate back to the `Shippping Configuration -> Shipping Rules` screen and click on the "Details" link next to the new rule you created. Then click the "Rule Ranges" link from the detail page, and the "Add New Record" button to create the first "rule range".

7. In our example, we'll define a package range for the big 60-mug box that our coffee mugs are packaged in by default. We'll set both the floor and the ceiling at 60. In addition, on this screen we'll enter the corresponding weight and dimensions of the box. After we add a range for the 60-mug box, we'll add another one for the 6-mug box. The result will look like this:

Table 6.1. Package Shipping Rule Ranges

Floor	Ceiling	Description	Weight
6	6	Small Box	3.4 lbs.
60	60	Large Box	40 lbs.

With this setup, if an order comes in for say 75 mugs, the shipping rule processor would create 1 package of weight 40, 2 packages of weight 3.4 lbs., and create another package for the final 3 mugs, using the weight as defined in the product record for the mug. (Anything in the basket that does not fit a package rule would simply be treated as a cumulative total weight package without dimensions, which is how shipping works when no shipping rules are defined.)

The shipping processor tries to fit the most items in a package first and then moves downward. It is important to understand that ONLY 1 package type shipping rule can be applied to a basket line item. Make sure to order your rules appropriately if you set up 2 different rules for SKUs that match the same line item.

The result is a very accurate rate quote from the UPS and USPS processors, because we're able to define with precision exactly how the mugs are shipped.

The UPSShippingProcessor will allow you to override certain global settings within a shipping rule using the pkgExtra 1-5 fields in a package rule range. For example, here are the UPS override fields:

1. Shipping Rule Range pkgExtra1 field = Override the default package code type
2. Shipping Rule Range pkgExtra2 field (0,1) = Override use package dimesions flag
3. Shipping Rule Range pkgExtra3 field (1,2,3) = Override oversized package code
4. Shipping Rule Range pkgExtra4 field (0,1) = Override use insured value flag

See the global UPS settings for more information on UPS settings.

Adding Method Limitations to Package Shipping Rules

Another reason for setting up package type rules is to limit shipping methods. In the above example, you might only be able to send the big 40 pound box via UPS Ground. In that case, you can define a method limitation within the "Rule Range" correcponding to the Large Box pacakage. As you create the Rule Range, in addition to defining the weight and dimensions of the large box, you would select "Only allow these methods" as the "Limit Shipping Method Type" field. For the "Limit Shipping Methods" field, you would then enter in "UPSShippingProcessor|02". (02 is the method code for Ground at UPS.) This tells the system to restrict the methods available for customers to select to just UPS Ground.

You can also choose a "Limit Shipping Method Type" of "Allow all but these methods," which would allow everything but certain methods. For example, you could say that you can't ship UPS Express on those items by entering in "UPSShippingProcessor|01" for the "Limit Shipping Methods" field.

Note

In addition to the above examples, you can use wildcards when setting up shipping method limits. For example, you could allow or disallow "UPSShippingProcessor|*", meaning all UPS methods would be disallowed.

Shipping Discount Rules

The next type of shipping rule you can create are Shipping Discount Rules. This allows you to discount the customer's shipping cost a certain amount based on what SKUs are in the cart, and in what quantities.

Follow the sames steps to create a Shipping Discount Rule as you did for a Package Shipping Rule in the example above, except select "Shipping Discount Rule" as the "Type of Shipping Rule". When you define the "Rule Ranges", instead of identifying a package weight and dimensions, you can define a percentage or fixed discount. If the user's cart contains one or more of the assigned SKUs and meets the quantity range for the "Rule Range", the discount will be applied against the order's shipping cost.

Note

In setting up the "Rule Ranges" for a Shipping Discount Rule, you must enter the shipping methods to apply the discounts to. If a discount would apply to all shipping methods, simply use the wildcard "*|*".

If a Shipping Discount Rule is applicable for user's order, the user is shown the original shipping price and the discount shipping price on the checkout screens when choosing a shipping method. They are also shown the names of any discounts that have been applied. This is a nice way to show the user what is happening. For instance they might see:

UPS: Ground

Discount: High Quantity Product X Shipping Discount

Price: $12.00

Original Price: $34.00

The rule processor will apply all possible discounts that match against the shipping totals until those methods hit $0. The user could see multiple discount lines above if multiple discounts were found to match in their basket for that method.

Note

If you setup a discount for a certain SKU that takes 100% off the total shipping price, the user could still be ordering many other things in their basket. The discount comes directly off the *total* shipping amount. So, if you give free shipping when you buy 12 product A's, it will still be free if they also had a bunch of other products in their basket.

'Method Limits by SKU' Shipping Rules

The next type of shipping rule you can define are those that limit shipping methods for particular SKUs, independently of any package definitions. Follow the sames steps to create a 'Method Limits by SKU' rule as you did for a Package Shipping Rule in the example above, except select "Method Limit Rule" as the "Type of Shipping Rule". Associate the SKUs you want to restrict methods for, and then create one

"Rule Range". Both the floor and the ceiling of the range can be 0, to indicate all quantities. Lastly, enter the method limitations for the range, as described above under Package Shipping Rules.

'Method Limits by Location' Shipping Rules

The last type of shipping rule you can define are those that limit shipping methods not for particular SKUs, but based on the location of the delivery address for the order. Many times, you want to restrict certain methods from appearing to international users, for example, or users outside the continental U.S.

Follow the sames steps to create a 'Method Limits by Location' rule as you did for a Package Shipping Rule in the example above, except select "Method Limit by Location Rule" as the "Type of Shipping Rule". There is no need to associate SKUs with this rule, since the rule is applied independently of what items are in the user's cart. Instead, simply create a "Rule Range" for the rule (in this case "Rule Range" is a bit of a misnomer). For this rule, highlight which countries and states the rule applies to. These are matched against the user's delivery address so that if the user's country or state matches the selected coutry or state, the rule will be applied. Lastly, enter the method limitations for the range, as described above under Package Shipping Rules.

Note

When defining the states and countries for a particular 'Method Limits by Location' rule, keep in mind that the rule will be applied if there is a match on *either* the state or the country. In other words, if you want a rule to apply just to customers in Alaska and Hawaii, do *not* select "United States" for the country because that will apply the rule to everyone in the U.S. Leave the country selection alone and just select Alaska and Hawaii from the state selection box.

Another Package Shipping Rule Example

Some stores selling very large products must ship a single product in multiple packages. For example, for large equipment, the product might need to assembled from multiple parts each of which is packaged and shipped in a separate box. The Shipping Rules function allows for this kind of set up as well. Let's illustrate this with another example.

Example 6.6. Defining Shipping Rules When One Product Is Shipped in Multiple Packages

1. Set up a Shipping Rule for each package that is part of the product's shipment. Under `Shipping Configuration -> Shipping Rules`, add a new Shipping Rule whose Type is "Package Size or Weight". Just for reference, give the new rule a code and a name and click the "Add New Record" button to save it.

2. Set up separate Shipping Rules for each package that the product is shipped in. All of them are "Package Size or Weight" rules.

3. Under each separate rule, add a new "Shipping Rule Range". Navigate to `Shipping Configuration -> Shipping Rules -> Details -> Rule Ranges`. In this case, we want to define the package regardless of how many of the product are being sold, so we set the Floor to 1 and the Ceiling to something very high such as 9999999.

4. As you set up the Rule Range, enter in the weight and dimensions for the package. This is what we'll use when sending the request to UPS.

5. For the Rule Range of the first Rule, the system will add the weights defined for the product, attrib-

utes and options. This is a good way to handle situations where there is variability for one of the packages of the shipment based on the options chosen by the customer. You might consider setting the weight of the first Rule Range to 0 and let the system use the product weight only instead.

6. Finally, we'll define which products the new rule applies to. From the shipping rule details screen, click the "SKUs" link.

7. Package shipping rules are associated with SKUs as opposed to products themselves. If you have not yet created a SKU record for the product, you would do so now by running the "SKU Builder" at the bottom of the SKU control screen by selecting the "Build All Product SKUs" option.

8. With the SKU created, assign it now to the shipping rule by clicking the "Edit Mode" to "on" on the SKU control screen, and checking off the "Assigned" checkbox for the SKU.

With this setup, if an order comes in for the product, the definitions for three packages will be sent to UPS (or USPS) when requesting rates.

Chapter 7. Configuring Payments

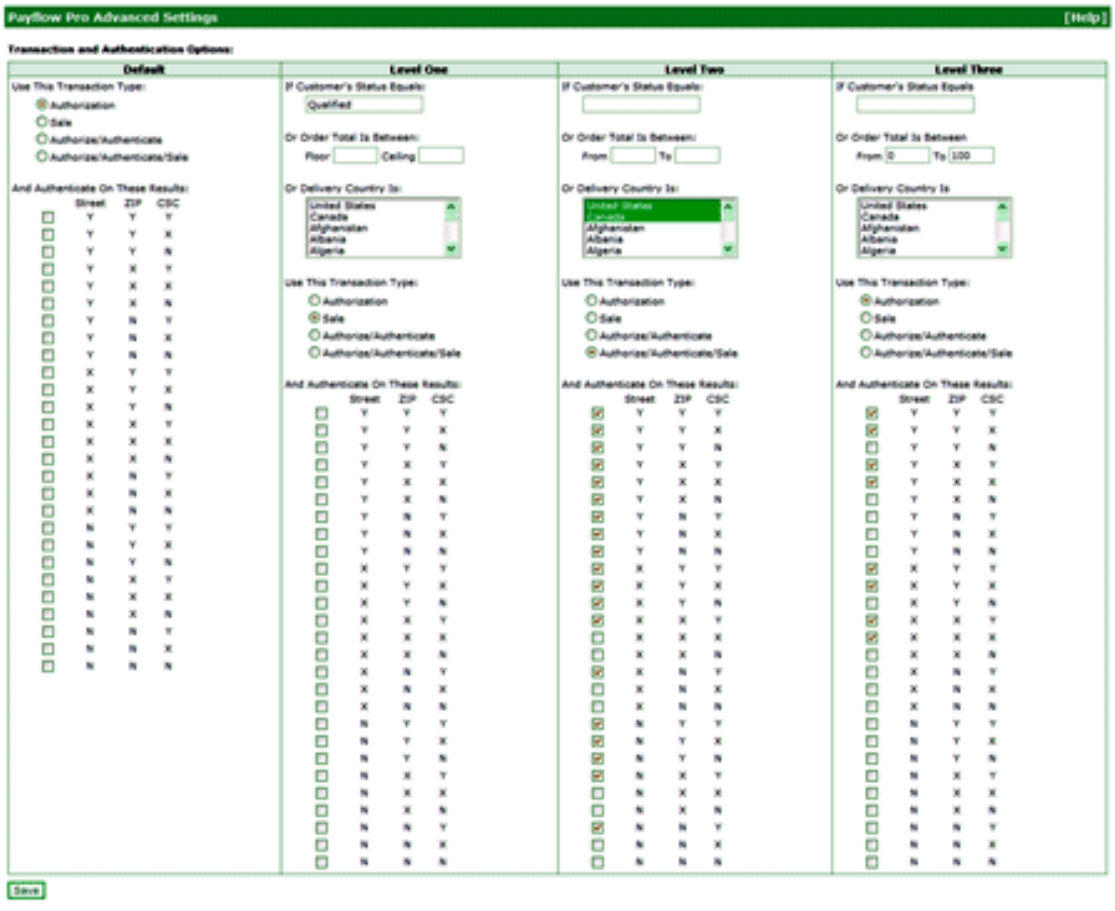
SoftSlate Commerce is distributed with several types of payment processors. The Basic Payment Processor allows you to capture credit card information for each order. When combined with two-way encryption (see the `Settings -> Security` screen), you can store the information in the database in encrypted format, and later retrieve it from the Payments screen for each order under the `Orders and Customers -> Orders` screen. This is useful if your store needs to process credit cards through a separate, offline system.

SoftSlate Commerce also integrates with PayPal's Payflow Link and Payflow Pro products to provide online credit card processing in real time. You must establish an account with PayPal and enter login information into the `Payment Configuration -> Payflow Link` or `Payment Configuration -> Payflow Pro` settings screens for these processors to work.

Note

Several of the fields on the `Payment Configuration -> Payflow Link` and `Payment Configuration -> Payflow Pro` screens accept placeholders that are replaced with dynamic values for each customer's order. For a list of accepted placeholders, refer to the Template Placeholders Reference section.

Payflow Pro Advanced Settings



Payflow Pro Advanced Settings screen. Larger Image [images/payflowProAdvancedLarge.gif].

The Payflow Pro Processor provides a number of advanced settings allowing you to tailor your online processing to your business, on the `Payment Configuration -> Payflow Pro Advanced Settings` screen. To explain how these advanced settings work, here is a step-by-step guide for how the Payflow Pro Processor handles each order as it is submitted by a customer.

1. The Payflow Pro Processor starts by comparing the customer's order with the settings defined under "Level One" of the Payflow Pro Advanced Settings:

   - *If Customer's Status Equals:* If the customer placing the order is logged in and has a "Status" corresponding to this value, the Level One settings will be applied to the order. All customers' status values can be modified at any time on the `Orders and Customers -> Customers` screen. For example, you may want to identify customers who have ordered in the past and label them with a status of "Qualified". When these customers come through to place a new order, you can automatically capture the order by setting the transaction type to "Sale" for their customer status. On the other hand, for all other customers, or for users who are placing an order without an account, you may want to authorize and authenticate their purchases first.

   - *If Order Total Is Between:* If there is no match on the customer's status, but the total price of the items in the current order falls within this range, the Level One settings will be applied to the order. This allows you to treat small orders, which may not be of great concern in terms of fraud, differently than large orders, which may require vetting. You could allow orders under, say, $100 to be captured automatically while larger orders would be authorized only, pending a manual review.

   - *Delivery Country Is:* If there is no match on either the customer's status or the order total, but the delivery address's country matches a selection in this drop down menu, the Level One settings will be applied to the order. The idea is you can treat orders going to different countries differently. Some countries may be a constant source of fraudulent orders, and others may not support AVS or card security code validations. In these cases, you can use this setting to identify those countries you wish to treat differently.

2. If none of the above settings under "Level One" applies, the processor next looks for a match with the "Level Two" settings, and, failing that, a match with the "Level Three" settings. If no match can be found, the processor uses the "Transaction Type" and "Authenticate On These Results" settings under the "Default" column.

3. Having determined which level of settings the given customer falls under, the processor next identifies the Transaction Type to be used for the order:

   - *Authorization:* If this option is selected under the level of settings that is in effect for the customer, an authorization request is sent to Payflow Pro, which places a hold on the customer's credit card account matching the amount of the order. You must capture the funds at a later time.

   - *Sale:* If this option is selected, the funds are immediately captured from the customer's credit card. There is no need to perform a delayed capture at a later time.

   - *Authorize/Authenticate:* First, the processor sends Payflow Pro an authorization request, which places a hold on the customer's credit card account matching the amount of the order. Next, the AVS and card security code results that come back from the authorization are tested against the checkboxes under the "Authenticate On These Results" settings. If the results are acceptable, the customer is considered "Authenticated" and the order is allowed to go through. If the results do not match one of the authentication checkboxes, the transaction is considered declined. An error message is given to the customer on the credit card payment screen asking him or her to double check the billing address and security code he or she has entered.

- *Authorize/Authenticate/Sale:* First, the processor sends Payflow Pro an authorization request, which places a hold on the customer's credit card account matching the amount of the order. Next, the AVS and card security code results that come back from the authorization are tested against the checkboxes under the "Authenticate On These Results" settings. If the results are acceptable, the customer is considered "Authenticated" and *an immediate Sale request is sent to Payflow Pro, capturing the funds that were just authorized*. If the results do not match one of the authentication checkboxes, the transaction is considered declined. An error message is given to the customer on the credit card payment screen asking him or her to double check the billing address and security code he or she has entered.

4. For the "Authorize/Authenticate" and "Authorize/Authenticate/Sale" transaction types, the processor checks the returned values for AVS Street, AVS ZIP, and card security code against the list of checkboxes. Note that a "Y" indicates a successful match for the street address, ZIP code, or security code. A "N" indicates a mismatch, and an "X" indicates that the customer's bank does not support that type of validation.

Chapter 8. Template Placeholders

Template Placeholders Reference

Several settings in the Administrator provide templates where the following placeholders can be used where dynamic data is meant to go. Please refer to the following table when constructing the templates.

Table 8.1. Template Placeholders Reference

Template Placeholder/Token	Description	Java Expression
%%LOGIN_URL%%	The URL used by a customer to log into his or her account. Useful for the Lost Password Email templates.	`(String) baseForm.getSettingsBean().getValue("customerSecureURL") + "/Account.do"`
%%STORE_NAME%%	The name of the store as defined in the `Settings -> Store` screen.	`(String) baseForm.getSettingsBean().getValue("storeName")`
%%STORE_ADDRESS1%%	The first line of the store's address, as defined in the `Settings -> Store` screen.	`(String) baseForm.getSettingsBean().getValue("storeAddress1")`
%%STORE_ADDRESS2%%	The second line of the store's address, as defined in the `Settings -> Store` screen.	`(String) baseForm.getSettingsBean().getValue("storeAddress2")`
%%STORE_CITY%%	The store's city, as defined in the `Settings -> Store` screen.	`(String) baseForm.getSettingsBean().getValue("storeCity")`
%%STORE_STATE%%	The store's state or province, as defined in the `Settings -> Store` screen.	`(String) baseForm.getSettingsBean().getValue("storeStateOrProvince")`
%%STORE_POSTALCODE%%	The store's postal code, as defined in the `Settings -> Store` screen.	`(String) baseForm.getSettingsBean().getValue("storePostalCode")`
%%STORE_COUNTRY%%	The store's country, as defined in the `Settings -> Store` screen.	`(String) baseForm.getSettingsBean().getValue("storeCountry")`
%%STORE_PHONE%%	The store's phone number, as defined in the `Settings -> Store` screen.	`(String) baseForm.getSettingsBean().getValue("storePhone")`
%%STORE_EMAIL%%	The store's email address, as defined in the `Settings -> Store` screen.	`(String) baseForm.getSettingsBean().getValue("storeEmail")`
%%STORE_FAX%%	The store's fax number, as	`(String) base-`

Template Placeholder/Token	Description	Java Expression
	defined in the Settings -> Store screen.	Form.getSettingsBean().getValue("storeFax")
%%ORDER_ID%%	The orderID from the sscOrder database table for the current userw's order, if it is defined.	Integer.toString(activeUser.getOrder().getOrderID())
%%CUSTOMER_ID%%	The customerID from the sscOrder database table for the current user's order, if it is defined.	Integer.toString(activeUser.getOrder().getCustomerID())
%%USER_NAME%%	The userName from the sscOrder database table for the current user's order, if it is defined.	activeUser.getOrder().getUserName()
%%ORDER_NUMBER%%	The orderNumber from the sscOrder database table for the current user's order, if it is defined.	Integer.toString(activeUser.getOrder().getOrderNumber())
%%ORDER_TOTAL%%	The formatted order total for the current user's order, if it is defined.	activeUser.getOrder().getFormattedTotal()
%%SHIPPING%%	The formatted shipping amount for the current user's order, if it is defined.	activeUser.getOrder().getFormattedShipping()
%%TAX%%	The formatted tax amount for the current user's order, if it is defined.	activeUser.getOrder().getFormattedTax()
%%ORDER_SUBTOTAL%%	The formatted subtotal (total minus tax and shipping) amount for the current user's order, if it is defined.	activeUser.getOrder().getFormattedSubtotal()
%%ORDER_TAXABLESUBTOTAL%%	The formatted taxable subtotal amount for the current user's order, if it is defined.	activeUser.getOrder().getFormattedTaxableSubtotal()
%%ORDER_DATE%%	The date the current user's order was completed.	activeUser.getOrder().getFormattedCompleted()
%%ORDER_LASTMODIFIED%%	The date the current user's order was last modified.	activeUser.getOrder().getFormattedLastModified()
%%ORDER_STATUS%%	The status from the sscOrder database table for the current user's order, if it is defined.	activeUser.getOrder().getStatus()
%%ORDER_STATUSDETAILS%%	The statusDetails from the sscOrder database table for the current user's order, if it is defined.	activeUser.getOrder().getStatusDetails()
%%ORDER_TRACKINGNUMBER%%	The trackingNumber from the sscOrder database table for the current user's order, if it is	activeUser.getOrder().getTrackingNumber()

Template Placeholder/Token	Description	Java Expression
	defined.	
%%ORDER_WEIGHT%%	The total weight from the sscOrder database table for the current user's order, if it is defined.	Double.toString(activeUser.getOrder().getWeight())
%%ORDER_QUANTITY%%	The total quantity of items from the sscOrder database table for the current user's order, if it is defined.	Double.toString(activeUser.getOrder().getQuantity())
%%BILL_FIRSTNAME%%	The first name of the billing address for the current user's order, if it is defined.	activeUser.getOrder().getFirstName()
%%BILL_LASTNAME%%	The last name of the billing address for the current user's order, if it is defined.	activeUser.getOrder().getLastName()
%%BILL_ORGANIZATION%%	The organization name of the billing address for the current user's order, if it is defined.	activeUser.getOrder().getOrganization()
%%BILL_ADDRESS1%%	The first line of the billing address for the current user's order, if it is defined.	activeUser.getOrder().getAddress1()
%%BILL_ADDRESS2%%	The second line of the billing address for the current user's order, if it is defined.	activeUser.getOrder().getAddress2()
%%BILL_CITY%%	The city of the billing address for the current user's order, if it is defined.	activeUser.getOrder().getCity()
%%BILL_STATE%%	The state or province of the billing address for the current user's order, if it is defined.	activeUser.getOrder().getState()
%%BILL_POSTALCODE%%	The postal code of the billing address for the current user's order, if it is defined.	activeUser.getOrder().getPostalCode()
%%BILL_COUNTRY%%	The country of the billing address for the current user's order, if it is defined.	activeUser.getOrder().getCountry()
%%BILL_PHONE1%%	The daytime phone number of the billing address for the current user's order, if it is defined.	activeUser.getOrder().getPhone1()
%%BILL_PHONE2%%	The evening phone number of the billing address for the current user's order, if it is defined.	activeUser.getOrder().getPhone2()
%%BILL_EMAIL1%%	The primary email address of the billing address for the current user's order, if it is defined.	activeUser.getOrder().getEmail1()
%%BILL_EMAIL2%%	The secondary email address of the billing address for the current user's order, if it is defined.	activeUser.getOrder().getEmail2()
%%BILL_EXTRA1%%	The extra1 field from the sscOrder database table for the current user's order, if it is	activeUser.getOrder().getExtra1()

Template Placeholder/Token	Description	Java Expression
	defined.	
%%BILL_EXTRA2%%	The extra2 field from the sscOrder database table for the current user's order, if it is defined.	activeUser.getOrder().getExtra2()
%%BILL_NOTES%%	The notes field from the sscOrder database table for the current user's order, if it is defined.	activeUser.getOrder().getNotes()
%%DELIVERY_ID%%	The deliveryID from the sscOrderDelivery database table for the current user's delivery record, if it is defined.	Integer.toString(activeUser.getOrder().firstDelivery().getOrderDeliveryID())
%%DELIVERY_TOTAL%%	The formatted total for the current user's primary delivery, if it is defined.	activeUser.getOrder().firstDelivery().getFormattedTotal()
%%DELIVERY_SHIPPING%%	The formatted shipping amount for the current user's primary delivery, if it is defined.	activeUser.getOrder().firstDelivery().getFormattedShipping()
%%DELIVERY_TAX%%	The formatted tax amount for the current user's primary delivery, if it is defined.	activeUser.getOrder().firstDelivery().getFormattedTax()
%%DELIVERY_SUBTOTAL%%	The formatted subtotal (total minus tax and shipping) amount for the current user's primary delivery, if it is defined.	activeUser.getOrder().firstDelivery().getFormattedSubtotal()
%%DELIVERY_TAXABLESUBTOTAL%%	The formatted taxable subtotal amount for the current user's primary delivery, if it is defined.	activeUser.getOrder().firstDelivery().getFormattedTaxableSubtotal()
%%DELIVERY_DATE%%	The date the current user's primary delivery was created.	activeUser.getOrder().firstDelivery().getFormattedCreated()
%%DELIVERY_LASTMODIFIED%%	The date the current user's primary delivery was last modified.	activeUser.getOrder().firstDelivery().getFormattedLastModified()
%%DELIVERY_STATUS%%	The status from the sscOrderDelivery database table for the current user's primary delivery, if	activeUser.getOrder().firstDeliv-

Template Placeholder/Token	Description	Java Expression
	it is defined.	ery().getStatus()
%%DELIVERY_STATUSDETAILS%%	The statusDetails from the sscOrderDelivery database table for the current user's primary delivery, if it is defined.	activeUser.getOrder().firstDelivery().getStatusDetails()
%%DELIVERY_TRACKINGNUMBER%%	The trackingNumber from the sscOrderDelivery database table for the current user's primary delivery, if it is defined.	activeUser.getOrder().firstDelivery().getTrackingNumber()
%%DELIVERY_WEIGHT%%	The total weight from the sscOrderDelivery database table for the current user's primary delivery, if it is defined.	Double.toString(activeUser.getOrder().firstDelivery().getWeight())
%%DELIVERY_QUANTITY%%	The quantity of items from the sscOrderDelivery database table for the current user's primary delivery, if it is defined.	Double.toString(activeUser.getOrder().firstDelivery().getQuantity())
%%DELIVERY_FIRSTNAME%%	The first name of the delivery address for the current user's order, if it is defined.	activeUser.getOrder().firstDelivery().getFirstName()
%%DELIVERY_LASTNAME%%	The last name of the delivery address for the current user's order, if it is defined.	activeUser.getOrder().firstDelivery().getLastName()
%%DELIVERY_ORGANIZATION%%	The organization name of the delivery address for the current user's order, if it is defined.	activeUser.getOrder().firstDelivery().getOrganization()
%%DELIVERY_ADDRESS1%%	The first line of the delivery address for the current user's order, if it is defined.	activeUser.getOrder().firstDelivery().getAddress1()
%%DELIVERY_ADDRESS2%%	The second line of the delivery address for the current user's order, if it is defined.	activeUser.getOrder().firstDelivery().getAddress2()
%%DELIVERY_CITY%%	The city of the delivery address for the current user's order, if it is defined.	activeUser.getOrder().firstDelivery().getCity()
%%DELIVERY_STATE%%	The state or province of the delivery address for the current user's order, if it is defined.	activeUser.getOrder().firstDelivery().getState()
%%DELIVERY_POSTALCODE%%	The postal code of the delivery address for the current user's order, if it is defined.	activeUser.getOrder().firstDelivery().getPostalCode()
%%DELIVERY_COUNTRY%%	The country of the delivery ad-	act-

Template Placeholder/Token	Description	Java Expression
	dress for the current user's order, if it is defined.	iveUser.getOrder().firstDeliv-ery().getCountry()
%%DELIVERY_PHONE1%%	The daytime phone number of the delivery address for the current user's order, if it is defined.	act-iveUser.getOrder().firstDeliv-ery().getPhone1()
%%DELIVERY_PHONE2%%	The evening phone number of the delivery address for the current user's order, if it is defined.	act-iveUser.getOrder().firstDeliv-ery().getPhone2()
%%DELIVERY_EMAIL1%%	The primary email address of the delivery address for the current user's order, if it is defined.	act-iveUser.getOrder().firstDeliv-ery().getEmail1()
%%DELIVERY_EMAIL2%%	The secondary email address of the delivery address for the current user's order, if it is defined.	act-iveUser.getOrder().firstDeliv-ery().getEmail2()
%%DELIVERY_EXTRA1%%	The extra1 field from the sscOrderDelivery database table for the current user's order, if it is defined.	act-iveUser.getOrder().firstDeliv-ery().getExtra1()
%%DELIVERY_EXTRA2%%	The extra2 field from the sscOrderDelivery database table for the current user's order, if it is defined.	act-iveUser.getOrder().firstDeliv-ery().getExtra2()
%%DELIVERY_NOTES%%	The notes field from the sscOrderDelivery database table for the current user's order, if it is defined.	act-iveUser.getOrder().firstDeliv-ery().getNotes()
%%START_ITEMS%%	Marks the beginning of a loop through the order items for the current user's order.	
%%CODE%%	The code of the order item currently being looped through.	orderItem.getCode()
%%NAME%%	The name of the order item currently being looped through.	orderItem.getName()
%%DESCRIPTION%%	The name of the order item currently being looped through.	orderItem.getName()
%%QUANTITY%%	The quantity of the order item currently being looped through.	order-Item.getQuantity()
%%COST%%	The unit cost of the order item currently being looped through.	order-Item.getFormattedCost()
%%PRICE%%	The unit price of the order item currently being looped through.	order-Item.getFormattedPrice()
%%ALT_PRICE%%	The alternate price of the order item currently being looped through.	order-Item.getFormattedAltPrice()

Template Placeholder/Token	Description	Java Expression
%%WEIGHT%%	The total weight of the order item currently being looped through.	`orderItem.getWeight()`
%%PRODUCT_WEIGHT%%	The unit weight of the order item currently being looped through.	`orderItem.getProductWeight()`
%%TOTAL%%	The total, extended price of the order item currently being looped through.	`orderItem.getFormattedTotal()`
%%SHORT_DESCRIPTION	The short description of the order item currently being looped through.	`orderItem.getShortDescription()`
%%EXTRA1%%	The value of the extra1 field of the order item currently being looped through.	`orderItem.getExtra1()`
%%EXTRA2%%	The value of the extra2 field of the order item currently being looped through.	`orderItem.getExtra2()`
%%EXTRA3%%	The value of the extra3 field of the order item currently being looped through.	`orderItem.getExtra3()`
%%EXTRA4%%	The value of the extra4 field of the order item currently being looped through.	`orderItem.getExtra4()`
%%EXTRA5%%	The value of the extra5 field of the order item currently being looped through.	`orderItem.getExtra5()`
%%START_ATTRIBUTES%%	Marks the beginning of a loop through the attributes for the current order item being looped through.	
%%ATTRIBUTE_CODE%%	The code of the current order item attribute being looped through.	`(String) attribute.get("attributeCode")`
%%ATTRIBUTE_NAME%%	The name of the current order item attribute being looped through.	`(String) attribute.get("attributeName")`
%%ATTRIBUTE_VALUE%%	The value chosen or entered by the user for the current order item attribute being looped through.	`(String) attribute.get("attributeValue")`
%%OIA_TOTAL%%	The total price for the current order item attribute being looped through.	`(String) attribute.get("total")`
%%OIA_WEIGHT%%	The total weight for the current order item attribute being looped through.	`(String) attribute.get("weight")`
%%ATTRIBUTE_COST%%	The unit cost of the order item attribute currently being looped through.	`(String) attribute.get("attributeCost")`
%%ATTRIBUTE_PRICE%%	The unit price of the order item attribute currently being looped through.	`(String) attribute.get("attributePrice")`

Template Placeholder/Token	Description	Java Expression
%%ATTRIBUTE_ALT_PRICE%%	The alternate price of the order attribute item currently being looped through.	(String) attribute.get("attributeAltPrice")
%%ATTRIBUTE_WEIGHT%%	The weight of the order item attribute currently being looped through.	(String) attribute.get("attributeWeight")
%%OPTION_CODE%%	The code of the option chosen by the user for the current order item attribute being looped through.	(String) attribute.get("optionCode")
%%OPTION_NAME%%	The name of the option chosen by the user for the current order item attribute being looped through.	(String) attribute.get("optionName")
%%OPTION_COST%%	The unit cost of the option chosen by the user for the order item attribute currently being looped through.	(String) attribute.get("optionCost")
%%OPTION_PRICE%%	The unit price of the option chosen by the user for the order item attribute currently being looped through.	(String) attribute.get("optionPrice")
%%OPTION_ALT_PRICE%%	The alternate price of the option chosen by the user for the order attribute item currently being looped through.	(String) attribute.get("optionAltPrice")
%%OPTION_WEIGHT%%	The weight of the option chosen by the user for the order item attribute currently being looped through.	(String) attribute.get("optionWeight")
%%END_ATTRIBUTES%%	Marks the end of a loop through the attributes for the current order item being looped through.	
%%END_ITEMS%%	Marks the end of a loop through the order items for the current user's order.	

Chapter 9. Upgrading SoftSlate Commerce

Upgrading Overview

The steps to upgrade SoftSlate Commerce from one version to another are the same regardless of the version you are upgrading from or to. The process of upgrading consists of first upgrading the application's files on the server, and then running the Upgrades tool in the Administrator to upgrade the data objects in the database.

To find out which version you are currently running, go to the `Maintenance and Upgrades -> Upgrades` screen of the Administrator. That screen tells you the current application version and data object version of the program. (You'll use this same screen to upgrade the database objects.)

- *Application Version:* The version of SoftSlate Commerce corresponding to the files residing on the server. (Specifically, this value is produced by the JSP templates used to produce the Administrator's screens.)
- *Data Objects Version:* The version of SoftSlate Commerce corresponding to the database objects, such as the table struture or the values of database records, for the database SoftSlate Commerce is connected to. (Specifically this value comes from the "dataObjectsVersion" record of the `sscSetting` table.)

Note

Unless you are in the middle of an upgrade, the application version and the data objects version should be exactly the same or unknown behavior may result, and a warning message will appear in the Administrator.

Important: Read This Before Upgrading

Before upgrading your copy of SoftSlate Commerce, please review the following details about how to handle any customizations you may have made to the application. If you have followed the Rules while making customizations to your installation of SoftSlate Commerce, in most cases an upgrade will preserve your changes without any manual intervention required. There are some exceptions, however.

Warning

Even if you have made your customizations using the techniques described in the Rules, there is still a small chance an upgrade will cause errors, since the files and code are by necessity interdependant to some degree. For this reason we strongly suggest running the upgrade in a test environment and placing some test orders before doing so on a production installation.

1. **Upgrading After Customizing Configuration Files and JSP Templates.**

   Many configuration files, and any custom JSP templates that are outside the `default` layout directory, are not replaced during the upgrade process. This allows you to preserve the customizations you have made while still leveraging changes to those files you have not customized. Specifically,

if your customizations have been limited to the following files and directories, installing an upgrade will not overwrite them.

Warning

If you have made changes to any other files besides the ones listed here *an upgrade will overwrite the modified files and eliminate your changes*. In these cases, please consider refactoring your changes into one of the following custom files.

Files and Directories Not Overwritten with Upgrades

- `/WEB-INF/classes/hibernate.properties` (database settings)
- `/WEB-INF/classes/log4j.properties` (logging settings)
- `/WEB-INF/classes/appSettings.properties` (general application settings)
- `/WEB-INF/conf/keys` (default location of encryption keys)
- `/WEB-INF/layouts/custom` (default location of custom JSP templates)
- All other directories under `/WEB-INF/layouts` other than `default`, which *is* overwritten.
- `/css/style-custom.css` (custom css stylesheet prior to 2.3.1)
- `/css/custom/custom.css` (custom css stylesheet for 2.3.1 and after)
- `/WEB-INF/conf/core/tiles-defs-pages.xml` (used for Tiles definitions of custom pages)
- All `struts-config-custom.xml` files under `/WEB-INF/conf` (used for custom Struts mappings)
- All `tiles-defs-custom.xml` files under `/WEB-INF/conf` (used for custom Tiles definitions)
- All `queries-custom.hbm.xml` files under `/WEB-INF/classes/resources` (used for custom HQL and SQL queries)
- All `application-custom.properties` files under `/WEB-INF/classes/resources` (used for custom messages)

2. **Upgrading After Customizing SoftSlate Commerce Source Code.**

   The best way to make source code changes to SoftSlate Commerce is to subclass one of the existing classes in the `com.softslate.commerce` Java package, and place the new class in your own Java package. Then, in the `/WEB-INF/classes/appComponents` file , you can update the implementing class name for the interface that is affected.

   It is possible, however, to modify the classes that come with SoftSlate Commerce and recompile them to make changes. If you have done this with a class in the `com.softslate.commerce` Java package, *the classes you have modified will be overwritten and replaced with the version of the class from the upgrade.* This pertains to both uncompiled source `.java` files in the `/WEB-INF/src` directory and compiled `.class` files in the `/WEB-INF/classes` directory.

   If this is the case, we suggest you refactor the changes you have made into a subclass as described above. Alternatively you can extract the upgrade and manually merge any changes that have taken

place to the class in question, compiling the merged class after you've made the merge.

3. **Upgrading from the Free Edition to the Standard Edition.**

   With one exception, the upgrade process described in the following section applies equally well to upgrades from the Free Edition to the Standard Edition, as it does for upgrades within the two editions. If you are upgrading from the Free Edition to the Standard Edition, the only additional step is to remove the `/WEB-INF/lib/softslate.jar` file from your installation before or after the upgrade process. The classes in the `softslate.jar` file will be replaced by `.class` files in the `/WEB-INF/classes` directory after the upgrade to the Standard Edition. This structure for the Standard Edition allows you to easily recompile the application from source code if necessary.

Upgrade Procedure

Note

Some of the following steps are illustrated using Tomcat 5 as the servlet container. If you are using a different application server, please identify the corresponding steps for the application server you are using before you begin.

1. **Prepare for the Upgrade.**

   a. *Make a Backup of the SoftSlate Commerce Application.* Make a back up copy of the current application directory. This is most important step of the entire process. If anything goes wrong you can restore the back up and investigate.

Important

Please don't skip this step! Making a backup copy of the working application directory is the most important step of the upgrade, because it means you can always go back to the current version of the application if anything goes wrong.

   b. *Review the Upgrade.* Review important release notes for the latest versions at http://www.softslate.com/documentation/html/releaseNotes.html [http://www.softslate.com/documentation/html/releaseNotes.html] . Examine the Change Log for the new features and bug fixes for the latest release at http://www.softslate.com/documentation/html/changeLog.html [http://www.softslate.com/documentation/html/changeLog.html] .

   c. *Download the Upgrade.* Download the SoftSlate Commerce upgrade from the URL given to you. If you don't know where to find the upgrade, contact sales or support. For Unix, the upgrade file will be a `.tar.gz` file and will look like this: `softslate-standardupgrade-x.x.x.tar.gz` or `softslate-compiledupgrade-x.x.x.tar.gz` . For Windows, the upgrade file will be `.zip` file and will look like this: `softslate-standardupgrade-x.x.x.zip` or `softslate-compiledupgrade-x.x.x.zip` .

   d. *Upload and Extract the Upgrade.* Upload the upgrade to your server and extract its contents into a working directory *outside the application server's area.* (Later, you will copy the contents of the upgrade into the SoftSlate Commerce directory, after bringing the application down.)

      For example, for a Unix platform, run the following command to untar the upgrade into a

working directory:

tar -zxf softslate-standardupgrade-x.x.x.tar.gz

A directory named `softslate` should appear with the contents of the upgrade inside it.

On Windows, right-click on the file and select "Extract All..." to unzip the file.

e. *Precompile New JSP Templates (Optional).* Optionally, you can precompile the new JSP templates for better performance after you restart the application. A build function for Tomcat 5 is provided to assist with this (SoftSlate Commerce Standard Edition only).

Use these steps to precompile on a Unix server (Tomcat 5 only):

i. **cd** into the `/WEB-INF/build` directory of SoftSlate Commerce.

ii. Run **./build.sh precompile -Dtomcat.home=<path to Tomcat installation> -Dpath.root=<path to working `softslate` directory>**. The compiled templates will be placed in the `/WEB-INF/classes/org/apache/jsp` directory. For example, if Tomcat is installed in the `/usr/local/apache-tomcat` directory and the working directory you've unpacked the files into is `/home/user/softslate` , you would run the following command:

./build.sh precompile -Dtomcat.home=/usr/local/apache-tomcat -Dpath.root=/home/user/softslate>

Note: It may take several minutes for the compilation to finish.

Use these steps to precompile on a Windows server (Tomcat 5 only):

i. Go to `Start -> All Programs -> Accessories -> Command Prompt` to open up a command prompt window.

ii. **cd** into the `/WEB-INF/build` directory of SoftSlate Commerce.

iii. Run **build.bat precompile -Dtomcat.home=<path to Tomcat installation> -Dpath.root=<path to working `softslate` directory>**. The compiled templates will be placed in the `/WEB-INF/classes/org/apache/jsp` directory. For example, if Tomcat is installed in the `E:\apache-tomcat` directory and the working directory you've unpacked the files into is `E:\softslate` , you would run the following command:

build.bat precompile -Dtomcat.home=E:\apache-tomcat -Dpath.root=E:\softslate>

Note: It may take several minutes for the compilation to finish.

f. *Create a Temporary HTML Page.* You should expect the following steps to take at least 10 minutes to complete, during which time SoftSlate Commerce will be unreachable. Consider creating an HTML page explaining to customers that the store will be down for maintenance. This file can be placed in the Web server so that when the application is brought down everyone will see it.

2. **Bring Down the SoftSlate Commerce Web Application.**

a. *Stop the SoftSlate Commerce Web Application.* For example, if you are using Tomcat use the Manager application to run the stop command:

```
http://localhost:8080/manager/stop?path=/softslate
```

b. *Test the Temporary HTML Page.* If you have created a static HTML page to inform users the store is down, verify that it is in place and working.

3. **Copy the Upgrade Over the Existing Application Directory.**

- *Unix.* Next, copy the contents of the upgrade you extracted into the application directory, replacing all existing files of the same name.

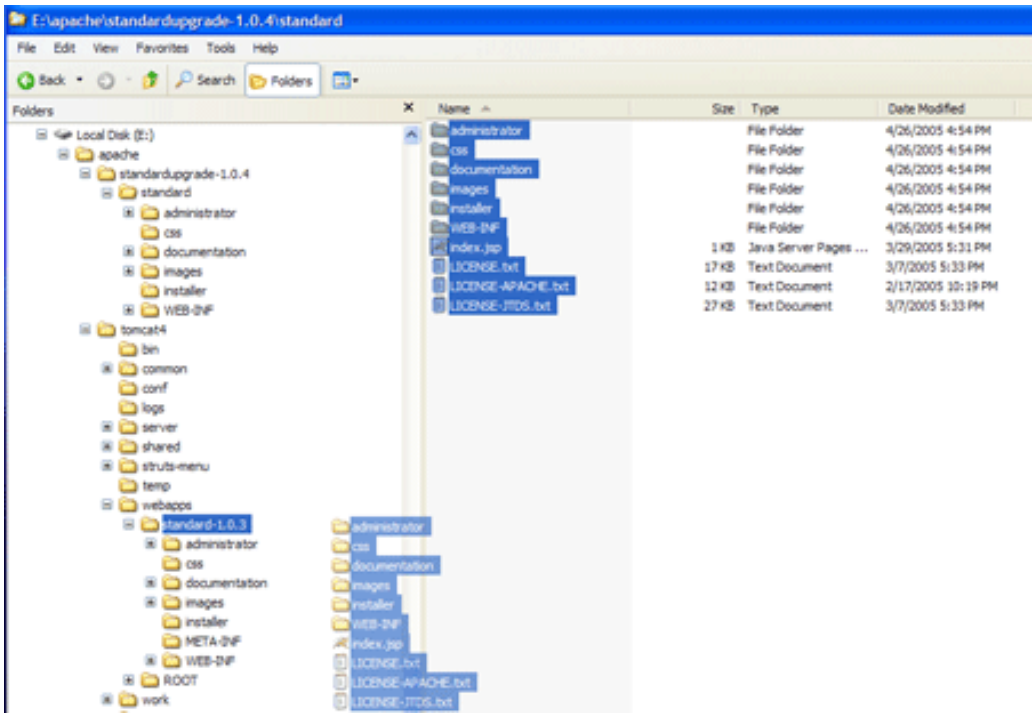
  On Unix use **cp -R** to perform the copy. This will replace the contents of the application directory with the contents of the upgrade directory. For example, if you extracted the upgrade into `/home/myuser`, and the application directory is at `/usr/local/apache-tomcat/webapps/softslate` , you would run this command to perform the copy:

  cp -R --reply=yes /home/myuser/softslate/* /usr/local/apache-tomcat/webapps/softslate

  After the copy, if necessary, you should **chown** the application directory to the correct owner and group:

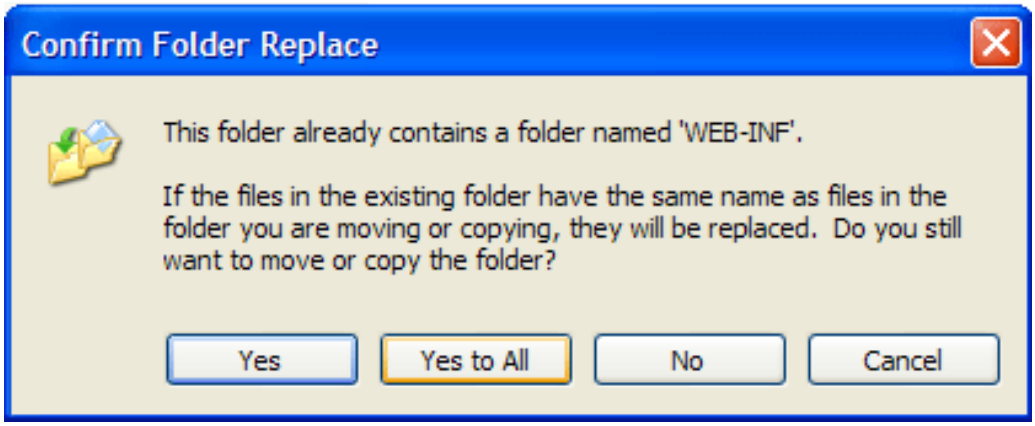
  chown -R tomcat.tomcat /usr/local/apache-tomcat/webapps/softslate

- *Windows.* On Windows use Windows Explorer to perform the copy:



Copying an Upgrade Over an Existing Application on Windows

When prompted to replace the existing content, select "Yes to All":



4. **Replace Previously Compiled Templates.**

   a. *Delete Previously Compiled JSP Templates.* Delete all the previously compiled JSP templates for the SoftSlate Commerce application. For example, for Tomcat, if the Tomcat installation directory is `/usr/local/apache-tomcat` , you would run this command:

      rm -r /usr/local/apache-tomcat/work/Catalina/localhost/softslate/org/apache/jsp

   b. *Copy Over Precompiled Templates (Optional).* If you precompiled the new JSP templates (above), copy or move them in to place now.

      Use these steps to copy the precompiled templates on a Unix server (Tomcat 5 only):

      i. Copy the precompiled templates into the `work` under the Tomcat distribution:

         mv /usr/local/apache-tomcat/webapps/softslate/WEB-INF/classes/org/apache/jsp / usr/local/apache-tomcat/work/Catalina/localhost/softslate/org/apache

      ii. Change the ownership of the precompiled templates to the owner of the Tomcat process:

         chown -R tomcat.tomcat / usr/local/apache-tomcat/work/Catalina/localhost/softslate/org/apache/jsp .

      On Windows, use Windows Explorer to copy the precompiled templates into the `work` directory under the Tomcat distribution. For example, for Tomcat, the compiled templates should go in this directory:

      `<tomcat-home>/work/Catalina/localhost/softslate/org/apache/jsp`

5. **Restart SoftSlate Commerce.**

   a. *Start the SoftSlate Commerce Application.* For example, if you are using Tomcat use the Manager application to run the start command:

      `http://localhost:8080/manager/start?path=/softslate`

6. **Upgrade the Database Objects.** Now that you've upgraded the application files for SoftSlate Commerce, you must upgrade the database objects that go with them. Until you do so, upgraded and fixed features will not work properly.

   a. *Log In to the Administrator.* Log in to the Administrator. You should see a warning message indicating that the application is out of sync with the database objects.

b. *Run the Uprades Tool.* Navigate to the `Maintenance and Upgrades -> Upgrades` screen, and click the "Run Upgrades" button to perform the database upgrades needed. If the upgrades are successful, you should see a success message and the warning should go away.

7. **Post-Upgrade Testing.**

a. *Place a Test Order.* Test the application by placing a test order and logging into the Administrator.

b. *Test the Upgraded Features.* Test the new features and bug fixes for the upgrade. For a list of the new and fixed features, visit http://www.softslate.com/documentation/html/changeLog.html [http://www.softslate.com/documentation/html/changeLog.html] .

Appendix A. Change Log

```
administrator 2.3.5 New SEO Code field added for products: it may be used instead
administrator 2.3.5 Better handling of image paths in the Froogle export
administrator 2.3.5 Text boxes on various admin screens made larger
administrator 2.3.5 Ability to validate administrator logins with image text verif
administrator 2.3.5 Lucene reindexing not triggered immediately after logging in a
administrator 2.3.5 Optimized the Lucene reindexing action to better handle tens o
all         2.3.5 Classes supporting the Lucene advanced search reorganized for a
all         2.3.5 Fixed bug where 'Illegal attempt' error thrown in rare cases whe
customer              2.3.5 Ability to validate new accounts with image text verifi
customer              2.3.5 Ability to validate credit card info with image text ve
customer              2.3.5 'Allow Auto Logins for Customers' option added to Setti
customer              2.3.5 'Automatically Save Last Cart' option added to Settings
customer              2.3.5 Remove button added next to each cart item on the cart
customer              2.3.5 Shipping options sorted by lowest price to highest on c
customer              2.3.5 Miscellaneous fixes and more thorough testing for PayPa
customer              2.3.4 Adding the ability to sort products by name when using
customer              2.3.4 UPS expected delivery date added to shipping options sc
customer              2.3.4 Bug fixed where Google Checkout failed when shipping is
customer              2.3.4 Optimized the product page when Lucene is being used to
customer              2.3.4 Fixed default-xhtml/core/rightSide.jsp to display corre
customer              2.3.4 Fixed bug where USPS rates failed to display for UK cus
administrator 2.3.4 Fix handling of Hibernate exceptions after constraint violatio
administrator 2.3.4 Added ability for the importer to process parent categories, a
all         2.3.4 Session IDs now logged on every hit at INFO level
all         2.3.3 Fixed bug in HibernateFilter where JSP exceptions could leave tr
administrator 2.3.3 Fix for Hibernate "Session is closed" exceptions after visitin
administrator 2.3.3 Removed width restrictions on SKU input select boxes
administrator 2.3.3 Removed unneeded Google Checkout jar files from build.xml file
administrator 2.3.2 New Product Sorter and Category Sorter functions for sorting a
administrator 2.3.2 Fixing formatting of the Output Records to ... Excel function
administrator 2.3.2 Fixed bug where control screen parameters were not properly re
administrator 2.3.2 All relevant child control screen parameters now reset when pa
all         2.3.2 Google Checkout enhanced to support order notifications and othe
all         2.3.2 Upgrade to the 0.91 Google Checkout reference implementation (Ja
all         2.3.2 Cleaned up classes and fixed numerous bugs related to the Google
all         2.3.2 Optimizing how the system loads matching SKUs for an order item
customer              2.3.1 New set of default JSPs named "default-xhtml" (used onl
customer              2.3.1 Several small fixes and improvements to the original "d
all         2.3.1 Upgrade to Hibernate 3.2.5
all         2.3.1 Upgrade to Xerces 2.9.1, eliminating errors with JDK 1.6 for UPS
all         2.3.1 Fixed bug in HibernateFilter where JSP exceptions could leave se
all         2.3.1 Unsupported JSP precompile and schema generation tasks taken out
installer             2.3.1 For security, all requests forward immediately to the i
installer             2.3.1 For security, the installer does not display database c
installer             2.3.1 Installer now detects up front what steps are completed
installer             2.3.1 Fixed bug in installer where the JDBC connection was le
installer             2.3.1 Look and feel of the installer updated
administrator 2.3.1 Bug in order export where date ranges were inaccurate fixed
administrator 2.3.1 When resetting the attribute control screen, the option screen
administrator 2.3.1 UPS extra weight field now added to individual packages in add
administrator 2.3.1 New UPS "maxPackageWeight" field allows admins to enforce UPS
administrator 2.3.1 New Display setting "Base Layout" defines which set of default
administrator 2.3.1 Ability to define logo, CSS theme, font style, and main colors
administrator 2.3.1 New Display setting named "Display Add to Cart on List Pages"
administrator 2.3.1 Legacy Display and Styles settings removed in new installation
administrator 2.3.1 Fixed bugs in roles framework for viewing payment information
documentation 2.3.1 Guide for Designers rewritten to describe CSS Themes and new D
administrator 2.2.6 Support added for importing primary categories and manufacture
```

```
administrator 2.2.6 Bug when price or weight is null during Lucene reindexing fixe
administrator 2.2.6 Preventing Administrators from being able to assign a category
installer            2.2.6 Missing Lucene index directory setting added to install
all         2.2.6 New "reloadSettingsHitCount" setting added to appSettings.proper
customer             2.2.6 Bug in the USPS Processor with incorrect country name f
customer             2.2.6 Bugs with display of category and product list pages wh
customer             2.2.6 Products not loaded from db for categories and manufact
customer             2.2.6 Fixing bug with font color of the order total in email
all         2.2.5 Lucene advanced search feature added
all         2.2.5 Added support classes and examples for running processing from t
all         2.2.5 Added new type of shipping rule, to restrict shipping methods ba
all         2.2.5 Fixed bug in the USPSProcessor so the 'is machinable' flag now w
all         2.2.5 Fixed bug in the PayflowProProcessor related to the AVS ZIP resp
all         2.2.5 Fixed bug in the BasePayPalNVPProcessor related to logging level
documentation 2.2.5 Documentation converted to new look and feel
documentation 2.2.5 Improved documentation for Shipping Rules
documentation 2.2.5 Documentation section on provisioning with RedHat and Tomcat a
customer             2.2.5 Added ability to display product attributes in a matrix
administrator 2.2.5 Added the ability to associate products with categories and ma
administrator 2.2.5 Added 'Extra Charge' and 'Extra Weight' fields to the UPS sett
administrator 2.2.5 Fixed bug in the order export so that the 'completed' field is
administrator 2.2.5 Fixed bug when using the resend order email function with a cu
administrator 2.2.4 Resend order emails function added to order details screen
administrator 2.2.4 Tax, shipping and payment roles modified to apply to relevant
all         2.2.4 USPS shipping integration
installer            2.2.4 PayPal US database settings now included in initial ins
all         2.2.3 More image file extensions added to those that SEOFilter ignores
all         2.2.3 Added a hook to the beginning of each request: ActionUtils.onPre
all         2.2.3 Fixed cases where HQL queries could not be overridden by appendi
all         2.2.3 Fixed bug in Google Checkout processing where incorrect order to
administrator 2.2.3 Fixed bug in User Defined Settings administrator screen affect
installer            2.2.3 Fixed bug affecting SQL Server 2005 in script that crea
administrator 2.2.2 User Defined Settings feature added under Settings -> User Def
administrator 2.2.2 Bug in Froogle export fixed related to the URLs for images.
administrator 2.2.2 Bug fixed when deleting a product that is another product's re
administrator 2.2.2 In the order export, the orderDelivery.statusDetails field has
administrator 2.2.2 Fixed spelling of role for Payment Configuration menu link are
administrator 2.2.2 Adding roles for order discounts, which were missing from the
all         2.2.2 Removing the call to initializeSQLProperties(), to prevent error
all         2.2.2 Fixes to Google Checkout to prevent errors during callback fucti
administrator 2.2.1 Language in the file downloads section clarified
administrator 2.2.1 Setting added to allow carts to be emptied upon customer log o
administrator 2.2.1 XML generation of Froogle and Google Sitemaps export changed t
administrator 2.2.1 Bug in Froogle export fixed related to when tax information wa
administrator 2.2.1 Bug in Authorize.net processor fixed where the po was overridd
administrator 2.2.1 Feature added for importing product information from flat text
administrator 2.2.1 Feature added for exporting product and order information to f
customer            2.2.1 PayPal Express Checkout link now shown on single-page c
customer            2.2.1 Fix for inappropriate message when editing cart and sel
customer            2.2.1 Fixed broken pages if an error occurs during registrati
customer            2.2.1 Fix for handling errors when invoking multiple shipping
customer            2.2.1 Populating the checkout address form from the current o
customer            2.2.1 "Other" suppressed from invoice display as state name
all         2.2.1 Caching entries added in ehcache.xml for manufacturers and shipp
all         2.2.1 Typo with hibernate.cache.use_second_level_cache in hibernate.pr
all         2.2.1 PayPal UK integration
all         2.2.1 Google checkout integration
all         2.2.1 UPS shipping integration
all         2.2.1 Shipping rules by SKU infrastructure added
all         2.2.1 Payflow Pro ACH support added
all         2.2.1 PayPal payment processors using common name-value pairs API refa
all         2.2.1 Making attribute -> option relationship all-delete-orphan in Hib
all         2.2.1 Logging of complete exception stack traces added throughout busi
customer            2.1.2 Deleting confusing application.properties files for tra
```

```
administrator 2.1.2 Fixing Froogle export format and required fields
administrator 2.1.1 Displaying component settings in alphabetical order in the adm
administrator 2.1.1 Fixed bug in Authorize.net processor where the test mode setti
administrator 2.1.1 Fixed bug where low stock emails not sent out if set to decrem
administrator 2.1.1 Changed sscOrderItemAttribute.attributeValue field to text dat
administrator 2.1.1 Added attributeValueMaxLength setting to enforce size of attri
customer             2.1.1 Checking inventory upon adding to cart even for items d
customer             2.1.1 If country allows 'Other' states, a state is not requir
customer             2.1.1 Handling null values in FormUtilsImpl
customer             2.1.1 Adding a SwitchLanguageAction which allows customers to
customer             2.1.1 Fixed bug with date display for some locales on credit
customer             2.1.1 Fixed bug where order total might not get updated after
customer             2.1.1 Added max file upload size of 10MB
customer             2.1.1 Validating CVV2 numbers to be 3 or 4 digits in length
customer             2.1.1 Improved error handling for shipping processors
customer             2.1.1 Add to cart processing now allows the same product with
customer             2.1.1 Fixed issue in PayflowProProcessor where ampersand and
all         2.1.1 Adjusted logging of admin classes to prevent exceptions if not a
all         2.1.1 Implemented PalPal Website Payments Pro (Express and Direct)
all         2.1.1 JSTL jstl.jar and standard.jar files updated to version 1.1.2
all         2.1.1 JavaMail mail.jar file updated to version 1.4
all         2.1.1 JAF activation.jar file updated to version 1.1
all         2.1.1 Manufacturers feature added
customer             2.0.11 Product, attribute, and option prices suppressed if ze
customer             2.0.11 Added error handling for shipping processors
customer             2.0.11 Added a /CartClear.do action which clears the cart of
customer             2.0.11 Fixed bug in query for the number of times a discount
customer             2.0.11 Fixed miniCategory.jsp template to display the All Ite
customer             2.0.11 Consolidated all JSPs displaying order details into la
customer             2.0.11 Moved findCheckoutScreen() from CheckoutAction to Acti
customer             2.0.11 Helper methods added to UserBean and BusinessObjectUti
administrator 2.0.11 Struts actions and mappings streamlined to make it easier to
administrator 2.0.11 Orphaned SKU records are now deleted when attributes and opti
administrator 2.0.11 Built-in category product count setting added under Settings
administrator 2.0.11 Administrators now remain logged in after placing an order as
administrator 2.0.11 Fixing failed login behavior for the administrator so a kinde
administrator 2.0.11 Restriction for discounts: if the times used is per customer,
customer             2.0.10 Fixed bug where multiple SKU discounts on the same ord
customer             2.0.10 Adding status details to the review order display, so
customer             2.0.10 Improving link to product zoom image to always pop up
customer             2.0.10 Fixed bug with shipping methods always showing to cust
customer             2.0.10 Reordering feature allows customers to reorder past or
customer             2.0.10 Saved cart feature allows customers to save carts and
customer             2.0.10 Fixed bug with addresses not being updated during chec
customer             2.0.10 Adjusting query for products in a category to fix bug
customer             2.0.10 SKU levels now being incremented when incomplete order
customer             2.0.10 Fixed bug with attribute weights being added to the or
customer             2.0.10 Fixed bug with one-way encrypted passwords being reset
customer             2.0.10 Fixed bug to force taxes to be reprocessed during chec
installer            2.0.10 Eliminating US Minor Outlying Islands from list of cou
customer             2.0.9 Fixing bug with taxable subtotal in BaseTaxProcessor wh
customer             2.0.9 Adding option to split checkout addresses and shipping
customer             2.0.9 SKU discounts with a fixed amount now expanded by the q
documentation 2.0.9 New sections on tracking inventory, and discounting in the Adm
documentation 2.0.9 New sections on extending the application in the Developer's g
customer             2.0.8 Struts actions nw using BaseForm whenever possible for
customer             2.0.8 Order history now paginated according to value of Order
customer             2.0.8 Pagination links on category screen now appear static w
customer             2.0.8 Fixed bug where customer delivery address was not being
customer             2.0.8 Forcing discounts, tax and shipping to be reprocessed o
customer             2.0.8 Missing forwards added to the OrderProcess Struts actio
administrator 2.0.8 Splitting out the definition of the application version from h
administrator 2.0.8 Fixed bug in BasicTaxProcessor where taxes for a state were ap
administrator 2.0.8 Tiles layouts enhanced for easier customization of admin templ
```

```
administrator 2.0.8 AuthorizeNetProcessor methods changed to protected so that the
administrator 2.0.8 Value fields in payment edit form expanded to maxlength 32
documentation 2.0.8 New chapters added to Developer's Guide
customer             2.0.7* Fixed NullPointerException as customers login without
documentation 2.0.7 API documentation updated from 1.x docs
customer             2.0.7 Hitslink interface now tracking thank you page in non-e
customer             2.0.7 More streamlined architecture in Struts customer accoun
customer             2.0.7 Processing discounts upon customer login and log out
customer             2.0.7 Fixed bug with SKU discounts not being added back into
administrator 2.0.7 Fixed breadcrumbing on Discount add screen
administrator 2.0.6 Authorize.net integration
administrator 2.0.6 Fixed bug with NullPointerException while editing from the SKU
customer             2.0.6 Redirecting back to cart screen if inventory adjustment
customer             2.0.6 Fixed bug with inventory processing at checkout if set
customer             2.0.6 Fixed bug with inventory messages not showing up during
installer            2.0.6 Default inventory decrement behavior changed to 'on ord
all         2.0.6 Missing jar files added to the compile classpath of the build.xm
all         2.0.5 Added the store address to the thank you screen, order details,
customer             2.0.5 Fixed bug where shipping was not reprocessed after rest
customer             2.0.5 Fixed password update bug when customer password encryp
customer             2.0.5 Coupon code removed from enteredCouponCodes session var
customer             2.0.5 Fixed bugs with flat rate shipping methods not working
customer             2.0.5 Fixed class cast issue in attributesAndOptions.jsp on R
customer             2.0.5 Attribute-only SKUs not loaded if inventory and discoun
customer             2.0.5 Fixed bug with the CheckPaymentProcessor and the orderi
customer             2.0.5 Non-serializable payment processor no longer stored in
administrator 2.0.5 Fixed bugs in the checkboxes of the shipping, payment, and tax
administrator 2.0.5 Password input type used for administrator passwords on the ad
administrator 2.0.5 Fixed issues with long load times on the SKU control screen
administrator 2.0.5 Fixed bug with the unique index for state codes so that they a
documentation 2.0.5 Updated installation and upgrade documentation
all         2.0.4 Implemented and tested enabling the Hibernate 2nd level cache fo
all         2.0.4 Implemented and tested enabling the Hibernate query cache for pr
all         2.0.4 Added a number of info log messages to make it easier to use the
customer             2.0.4 Fixed bug on product screen and CartAdd action when att
customer             2.0.4 Relevant discounts tied to attribute-only SKUs now disp
administrator 2.0.4 Fixed bug where running upgrades caused errors in Oracle
administrator 2.0.4 Text file import feature removed from menu due to bugs and dat
installer            2.0.4 Adding skus and a discount to the product test data
administrator 2.0.3 Fixed bug with null settings in the Dao factory immediately af
administrator 2.0.3 Fixed bug in 2.0.1 to 2.0.2 upgrade script regarding npcSettin
administrator 2.0.3 Upgrade process adapted to account for database table prefix s
administrator 2.0.3 Fixed bug where a primary category could not be set to none fr
administrator 2.0.3 Fixed bug where a parent category could not be set to none fro
administrator 2.0.3 Fixed bug where a tax rate's state could not be set to any fro
administrator 2.0.3 Fixed bugs regarding typos with admin roles in the administrat
administrator 2.0.3 Fixed bug where the built in categories setting could not be u
administrator 2.0.3 Fixed bugs regarding broken breadcrumbing links in the orders
administrator 2.0.3 Discount settings split out on to a separate screen
administrator 2.0.3 Fixed bug with products link from category control screen and
administrator 2.0.3 Feature to keep ordering fields in sync for product associatio
customer             2.0.3 Fixed pagination issues on category screen
customer             2.0.3 Fixed bug in style.css and style.jsp where a:hover was
customer             2.0.3 Fixed bug with broken category pagination links when an
customer             2.0.3 Discounting tested, and discounting display improved
customer             2.0.3 Coupon form and display settings added to cart screen,
customer             2.0.3 Fixed bug in PayflowProProcessor where not requiring a
documentation 2.0.2 Documentation made xhtml compliant
all         2.0.2 Default configurations for c3p0 adjusted to avoid mysql connecti
all         2.0.2 Upgraded to mysql driver version 3.1.12
customer             2.0.2 Fixed bug in BasicTaxProcessor when a tax rate only app
administrator 2.0.2 Fixed bug with SMTP authentication
administrator 2.0.2 Fixed bug when clicking Products link from category detail pag
administrator 2.0.1 SMTP user name and password added to allow for SMTP authentica
```

```
administrator 2.0.1 SKU builder added for bulk SKU creation
administrator 2.0.1 Text file imports of product information
administrator 2.0.1 Jump to page drop down added to all control screens
administrator 2.0.1 Delete and Assigned labels highlighted to check all rows at on
administrator 2.0.1 Logging in while already logged in no longer throws an error
administrator 2.0.1 List of built-in categories now a setting under System setting
administrator 2.0.1 Fixed bug on administrator control screen involving assignment
all         2.0.1 Integration with Hibernate data persistance framework
all         2.0.1 Discounting framework (not fully tested in 2.0.1)
all         2.0.1 Inventory tracking including administration, display and process
all         2.0.1 Related products including administration and display
all         2.0.1 Eliminated all Eclipse 3.1 compilation warnings
all         2.0.1 All beans made serializable for Hibernate lazy loading
all         2.0.1 Ability to sell virtual products as downloadable files
customer              2.0.1 Added setting to end session on order completion
customer              2.0.1 Product, category and other pages made XHTML compliant
customer              2.0.1 Explicitly setting sent date on all emails produced by
customer              2.0.1 Invoice and notification emails now use customizable JS
customer              2.0.1 Checkout login screen conformed to rest of checkout scr
customer              2.0.1 Logging in while already logged in no longer throws an
customer              2.0.1 Product and category not found errors no longer cause t
customer              2.0.1 Default productList.jsp switched to display products in
customer              2.0.1 Attributes, options, and inventory displayed in compact
customer              2.0.1 Attributes broken up into separate tiles dynamically in
customer              2.0.1 Several fixes and improvements made to attribute and op
customer              2.0.1 Breadcrumbing on product, category and other pages
customer              2.0.1 Search results and product list organized by primary ca
customer              2.0.1 Added extra order item fields, short description to ema
customer              2.0.1 Employed JSTL tags in JSPs to reduce amount of scriptle
customer              2.0.1 Facility to override MessageResources using -custom pro
installer             2.0.1 PostgreSQL support
installer             2.0.1 Oracle support
installer             2.0.1 Fixed compilation warnings regarding java.io.DataInputS
```

* Patched in 2.0.7 very shortly after its initial release.

Appendix B. Release Notes

Version 2.x

Migrating from 1.x to 2.x.

Because of a number of architectural changes, the upgrade process from 1.x to 2.x does not follow the usual automated procedure. For information on migrating from 1.x to 2.x, please review the Migration Guide .

Version 2.3.5

Interface Changes - Lucene classes.

The class structure supporting the Lucene and Advanced Search processing was reorganized to provide better functional separation and modularity. At the next restart, the system will attempt to add the following lines to your `/WEB-INF/classes/appComponents.properties` file to support the new interface implementations. If the server does not have writable permissions for that file, or if you are maintaining that file manually, you must add these configurations manually as you perform the upgrade.

```
luceneIndexerImplementer = com.softslate.commerce.businessobjects.product.BasicLuc
luceneSearcherImplementer = com.softslate.commerce.businessobjects.product.BasicLu
```

The previous property, "luceneProcessorImplementer" is no longer used.

If you have made customizations to the way Lucene indexes or searches, you will need to update your customizations to conform to the new class structure. The previous interface, `com.softslate.commerce.businessobjects.product.LuceneProcessor` has been replaced by two new interfaces: `com.softslate.commerce.businessobjects.product.LuceneIndexer` and `com.softslate.commerce.businessobjects.product.LuceneSearcher`. So if you have created your own implementation of `LuceneProcessor`, you should incorporate your customizations into implementations of one or both of the new interfaces. Specifically, if you have subclassed `BasicLuceneProcessor`, you should now subclass `BasicLuceneIndexer` and/or `BasicLuceneSearcher`, and update the values of the new "luceneIndexerImplementer" and "luceneSearcherImplementer" properties in `appComponents.properties` to identify your subclass.

Interface Changes - other.

The following properties have been added to the application interfaces to support new features. If you have created your own custom Java classes please take note of the following interface changes. If you have created an implementation of any of these interfaces, please update your implementations as you upgrade.

1. Implementations of `com.softslate.commerce.businessobjects.product.Product` and `com.softslate.commerce.businessobjects.order.OrderItem` must now implement `String getSeoCode()` and `void setSeoCode(String seoCode)`
2. Implementations of

`com.softslate.commerce.businessobjects.customer.Customer` must now implement `String getAutoLoginToken()` and `void setAutoLoginToken(String autoLoginToken)`

3. Implementations of `com.softslate.commerce.businessobjects.order.Order` must now implement `String getAutoSavedCartToken()` and `void setAutoSavedCartToken(String autoSavedCartToken)`

Version 2.3.2

Interface Additions.

Several interface additions were made to support new Google Checkout features. At the next restart, the system will attempt to add the following lines to your `/WEB-INF/classes/appComponents.properties` file to support the new interface implementations. If the server does not have writable permissions for that file, or if you are maintaining that file manually, you will want to add these configurations manually as you perform the upgrade.

```
googleRiskInformationNotificationProcessor= com.softslate.commerce.businessobjects
googleOrderStateChangeNotificationProcessor=com.softslate.commerce.businessobjects
googleChargeAmountNotificationProcessor= com.softslate.commerce.businessobjects.pa
googleRefundAmountNotificationProcessor= com.softslate.commerce.businessobjects.pa
googleChargebackAmountNotificationProcessor=com.softslate.commerce.businessobjects
googleAuthorizationAmountNotificationProcessor= com.softslate.commerce.businessobj
```

Google Checkout Libraries.

Version 2.3.2 upgrades the Google Checkout Reference implementation API library to version 0.91. As part of this, two of the three Google Checkout jar files are no longer required, and will actually prevent the Google Checkout feature from working. You must remove from the application's `/WEB-INF/lib` directory as you upgrade. The two files you must delete are: `JavaCheckoutApi.jar` and `JavaCheckoutExampleCommon.jar`

Version 2.3.1

hibernate.properties Changes.

Two manual changes may be required before upgrading to 2.3.1, due to the upgrade to Hibernate 3.2. Hibernate now enforces the existance of the `hibernate.cache.use_second_level_cache` property in `hibernate.properties`. Because versions prior to 2.2.1 shipped with a typo for that property, you should manually correct the typo in your `hibernate.properties` file before upgrading to 2.3.1. Specifically, if your `hibernate.properties` file has a line for this property:

```
hibernate.cglib.use_second_level_cache
```

you must correct it to read like this:

```
hibernate.cache.use_second_level_cache
```

In addition, if you have enabled Hibernate's second level cache or query cache, Hibernate now enforces the existance of the `hibernate.cache.provider_class` property in `hibernate.properties`. Because versions prior to 2.2.1 shipped without that property, you should manually add the property in your `hibernate.properties` file before upgrading to 2.3.1. Specifically, add the following line to your `hibernate.properties` file:

```
hibernate.cache.provider_class=org.hibernate.cache.EhCacheProvider
```

Settings Changes.

The `imageSystemPath` and `imagePathURL` settings have changed slightly in meaning. These settings only affect the image upload functions of the Administrator and no changes are typically necessary for upgrading. If the `imageSystemPath` setting is blank, the application will attempt to derive it from the `imagePathURL` setting (using `getRealPath()`). At the same, again only if `imageSystemPath` is blank, the system will prepend the application's installation directory to the `imagePathURL` setting when uploading and saving files.

xerces.jar.

Versions prior to 2.3.1 shipped with Xerces 1.4, consisting of the `/WEB-INF/lib/xerces.jar` library. Including this library in installations running on the 1.6 Sun JDK causes errors with the UPS and USPS processors, and with the Google Sitemaps and Froogle (Google Base) exports. To fix this, we have upgraded Xerces to 2.9.1, which consists of 5 other jar files. If you upgrade, it is recommended that you delete the old, now-redundant `xerces.jar` file. Our testing indicates that it will cause a problem if present on stores running with JDK 1.6.

New Default JSP Templates and the 'Base Layout' Setting.

Version 2.3.1 includes a completely reworked set of JSP templates featuring complete XHTML compliance and modern CSS practices. The new set of default templates is located in the `/WEB-INF/layouts/default-xhtml` directory.

However, if you are upgrading from a previous version, to maintain backwards compatibility, your store will continue to use the original set of default templates, in `/WEB-INF/layouts/default`. The result is you can install the upgrade without worry that your pages will break. We will continue to support and maintain the original default set of templates for the foreseeable future; however, at a future point in time we may elect to drop maintenance.

If you would like to take advantage of the new default templates, you can switch after upgrading. Under the `Settings -> Display` screen, modify the "Base Layout" setting to "default-xhtml". Doing so may break your pages, if you have created custom templates that interact with the default templates. We recommend you go through your custom templates as part of the process of switching, and rework them to conform to the HTML code of the new set of defaults.

Version 2.2.1

JSP Changes.

One key JSP change and some additions were made to the way the shipping options are displayed, in order to accomodate the new shipping rules functionality. If you have created a custom version of `/`

`WEB-INF/layouts/default/order/shippingFormGuts.jsp`, you must update it so it submits the shipping option in the new format.

1. In the JSP template that produces the shipping options displayed to the user, `/WEB-INF/layouts/default/order/shippingFormGuts.jsp`, the format of the form parameter for the shipping option that is submitted has changed. Where it used to submit simply the "code" of the shipping option, it now submits the processor class name and the code, in a pipe-delimited string: <processorclassname>|<code>. Refer to `/WEB-INF/layouts/default/order/shippingFormGuts.jsp` in the new upgrade for an example.

2. Additional map elements for each option are returned now with the built in shipping processors, which you can now use to display in the shipping selection form. These new map elements include: `processorClass`, `processorName`, `originalPriceDouble`, `originalPrice`, and `discounts`, which is a `Collection` of discounts applicable to the shipping cost.

Shipping Processor Changes.

Although no formal interface changes were made, the contract between the `com.softslate.commerce.businessobjects.shipping.ShippingProcessor` interface, the `com.softslate.commerce.businessobjects.shipping.BaseShippingProcessor` abstract class, and all of their implementations has changed. If you have created your own custom Java shipping processor class, please review your implementations as you upgrade your application to 2.2.

1. New elements are required to be present in the `Map` objects returned by the `Map loadShippingOptions(Map parameters)` method, which represent each shipping option:

   `priceDouble`: The raw price for the shipping option represented as a `Double`. This object is used to set the shipping amount in the order by `com.softslate.commerce.businessobjects.shipping.BaseShippingProcessor`. It is now required to be present in the `Map` objects returned by `Map loadShippingOptions(Map parameters)`.

   `processorClass`: The name of the class handling this option. If your implementation subclasses `com.softslate.commerce.businessobjects.shipping.BaseShippingProcessor`, this element will be inserted for you, with the class name of your processor. This is element is now used as part of the form parameter submitted with the value of the shipping option so the system can identify both the processor class and the option that was selected.

   `processorName`: The name displayed to users in front of each of the options, identifying the carrier or shipper for each option. If your implementation subclasses `com.softslate.commerce.businessobjects.shipping.BaseShippingProcessor`, this element will be inserted for you, with a string that is derived by taking the class name and eliminating the string "ShippingProcessor" from it. For example, the options for the `UPSShippingProcessor` will be displayed with "UPS" in front of each.

2. Whereas previously it was up to each shipping processor to ensure that the shipping option selected by the user was its own, now only the processor corresponding to the selected option is invoked.

   In addition to passing in the `shippingOption` selected by the user, a new map element, `availableOptions` is passed in to the `Map processShipping(Map parameters)` method. This `Collection` contains a filtered list of options that are available, according to any shipping rules that may be in effect. The processor's own `loadShippingOptions()` is called to produce the list, and then the applicable shipping rules are applied to the options.

`com.softslate.commerce.businessobjects.shipping.BaseShippingProcessor` calls new helper methods within its implementation of `Map processShipping(Map parameters)`. The new methods verify that the shipping selection is in the allowed list of `availableOptions`, and also set the shipping cost in the order. In many cases, the `Map processShipping(Map parameters)` of `BaseShippingProcessor` is all that is needed to process shipping for most processors. The abstract method `int processShipping()` is still available as a hook for custom shipping processors but in many cases it need only return a 0, for a successful processing.

Interface Additions.

Several interface additions were made to support new features. At the next restart, the system will attempt to add the following lines to your `/WEB-INF/classes/appComponents.properties` file to support the new interface implementations. If the server does not have writable permissions for that file, or if you are maintaining that file manually, you will want to add these configurations manually as you perform the upgrade.

```
googleNewOrderNotificationProcessor=com.softslate.commerce.businessobjects.payment
importDAOImplementer=com.softslate.commerce.daos.importexport.ImportDAOHibernate
importProcessorImplementer=com.softslate.commerce.businessobjects.importexport.Bas
exportProcessorImplementer=com.softslate.commerce.businessobjects.importexport.Bas
```

Interface Change.

One interface change was made to support the new image upload feature. Please take note of the following change snd update your custom implementation if necessary, as you upgrade your application to 2.2.1.

1. Implementations of `com.softslate.commerce.customer.core.ActionUtils` must now implement `Integer saveFile()` and `Integer makeFileNameNice()`

Key Settings Added.

Two new system settings controlling the location of uploaded images have been added, and these should be set manually before the new image upload feature is used. Go to the Settings -> System screen to define the URL path and the system path to the directory where uploaded images are placed.

Version 2.1.1

Interface Additions.

Several interface additions were made to support new manufacturers feature. At the next restart, the system will attempt to add the following lines to your `/WEB-INF/classes/appComponents.properties` file to support the new interface implementations. If the server does not have writable permissions for that file, or if you are maintaining that file manually, you will want to add these configurations manually as you perform the upgrade.

```
manufacturerProcessorImplementer =    com.softslate.commerce.businessobjects.produ
manufacturerImplementer =      com.softslate.commerce.businessobjects.product.Manu
```

```
manufacturerGatewayDAOImplementer =   com.softslate.commerce.daos.product.Manufact
manufacturerDAOImplementer =     com.softslate.commerce.daos.product.ManufacturerD
```

Interface Changes.

Several interface changes were made to support the new manufacturers feature. If you have created your own custom Java classes please take note of the following interface changes. If you have created an implementation of any of these interfaces, please update your implementations as you upgrade your application to 2.1.1.

1. Implementations of `com.softslate.commerce.businessobjects.product.Product` must now implement `Integer getManufacturerID()`
2. Implementations of `com.softslate.commerce.businessobjects.product.Product` must now implement `void setManufacturerID(Integer manufacturerID)`
3. Implementations of `com.softslate.commerce.businessobjects.product.Product` must now implement `Manufacturer getManufacturer()`
4. Implementations of `com.softslate.commerce.businessobjects.product.Product` must now implement `void setManufacturer(Manufacturer manufacturer)`
5. Implementations of `com.softslate.commerce.businessobjects.product.ProductProcessor` must now implement `Collection loadManufacturers()`

Hibernate Mapping File Changes.

If you have created your own custom Hibernate mapping file replacing `ProductBean.hbm.xml` please take note of the following additional mappings. Please add these mappings to your custom mapping file as you upgrade your application to 2.1.1.

1.

```
<property name="manufacturerID" insert="false" update="false"/>
```

2.

```
<many-to-one
   name="manufacturer"
   class="com.softslate.commerce.businessobjects.product.Manufacturer"
   column="manufacturerID"
   cascade="none"
   not-null="false"/>
```

Version 2.0.10

Interface Changes.

Several interface changes were made to support the new saved cart and reordering features. If you have created your own custom Java classes please take note of the following interface changes. If you have created an implementation of any of these interfaces, please update your implementations as you upgrade your application to 2.0.10.

1. Implementations of `com.softslate.commerce.businessobjects.customer.CustomerProcessor` must now implement `Collection loadSavedCarts()`
2. Implementations of `com.softslate.commerce.daos.customer.CustomerDAO` must now implement `Collection loadSavedCarts()`
3. Implementations of `com.softslate.commerce.businessobjects.order.CustomerProcessor` must now implement `Order loadSavedCart()`
4. Implementations of `com.softslate.commerce.daos.order.OrderDAO` must now implement `Order loadSavedCart()`
5. Implementations of `com.softslate.commerce.businessobjects.order.CustomerProcessor` must now implement `Map saveCart(Map)`
6. Implementations of `com.softslate.commerce.businessobjects.order.OrderProcessor` must now implement `Order duplicateOrder(Order)`
7. Implementations of `com.softslate.commerce.businessobjects.order.OrderProcessor` must now implement `Order reprocessOrder(Order)`
8. Implementations of `com.softslate.commerce.businessobjects.order.CustomerProcessor` must now implement `Order loadPreviousOrder()`
9. Implementations of `com.softslate.commerce.daos.order.OrderDAO` must now implement `Order loadPreviousOrder()`

Version 2.0.9

Changes to Checkout Screens.

Several changes were made to the store's checkout screens: namely, a new settings now exists that allows you to split the address form and the shipping option selection into two separate screens. (On the `Settings -> Logic` screen, set the "Use Combined Address and Shipping Screen in Checkout?" setting to no.)

If you have made any customizations to the Struts actions, action forms, or the Struts configurations involved with the checkout screens, particularly the `/CheckoutAddresses.do` screen, please note the following changes: following:

1. The Struts action mapping for the `/CheckoutAddresses` path in the `/WEB-INF/conf/customer/struts-config-order.xml` file now corresponds to a new checkout screen which only promts the user to enter in address information (not a shipping option).

2. The action form bean `com.softslate.commerce.customer.order.CheckoutAddressesForm` no longer holds the shipping option selection and no longer validates it.

3. The action class `com.softslate.commerce.customer.order.CheckoutAddressesAction` and the `productCheckoutAddresses` method in `com.softslate.commerce.customer.order.CheckoutAction` no longer processes shipping.

4. The combined address and shipping screen is now handled by the Struts action mapping for the `/CheckoutComboShipping` path in the `/WEB-INF/conf/customer/struts-config-order.xml` file, and the corresponding form class `com.softslate.commerce.customer.order.CheckoutComboShippingForm` and action class `com.softslate.commerce.customer.order.CheckoutComboShippingAction` .

5. The JSP file `/WEB-INF/layouts/default/order/checkoutBreadcrumbs.jsp` has been completely redone to take into account the new screens. If you have created a custom version of this file, please make sure the customizations work correctly after upgrading.

Version 2.0.7

Changes to Customer Accounts Screens.

Numerous changes were made to the customer accounts section of the store, to make the code and the configurations more streamlined and easier to customize. The changes were limited to the following:

1. The Struts action forms and action classes in the `com.softslate.commerce.customer.customer` package.

2. The Struts configurations in the `/WEB-INF/conf/customer/struts-config-customer.xml` file.

If you have created custom Struts configurations that override the configurations defined in the `/WEB-INF/conf/customer/struts-config-customer.xml` file, you must review the new code and incorporate your changes into it, as you upgrade to version 2.0.7.

Interface Changes.

Two interface changes were made to the `com.softslate.commerce.businessobjects.order.CartDiscountProcessor` interface. If you have created your own custom Java class implementing this interface, please take note of the following two new additional methods that were added to it. (If your custom class is a subclass of `com.softslate.commerce.businessobjects.order.BasicCartDiscountProcessor` , you may not need to make any changes as it will employ that class's implementation.)

1. `Map processCustomerLogin(Map parameters)` has been added to the interface.

2. `Map processCustomerLogout(Map parameters)` has been added to the interface.

Version 2.0.5

Interface Change.

Please note the following API change. If you have created an implementation of the following interface, please update your implementation as you upgrade your application to 2.0.5.

`CustomerGatewayDAO.updateCustomer(Customer customer)` now takes an additional boolean forcePassword argument, which tells the application to force the password field to be updated (as in the case of a customer losing the password and wishing it to be reset.

Version 1.0.11

Interface Changes.

Several interface changes were made to make the API more flexible. If you have created your own custom Java classes please take note of the following interface changes. If you have created an implementation of any of these interfaces, please update your implementations as you upgrade your application to 1.0.11.

1. `AdministratorProcessor.processLogin` now returns a Map rather than void
2. `CustomerProcessor.processRegister` now returns a Map rather than void
3. `CustomerProcessor.processLogin` now returns a Map rather than void
4. `CustomerProcessor.updateAddresses` now returns a Map rather than void
5. `CustomerProcessor.updateCustomer` now returns a Map rather than void
6. `CustomerProcessor.loadOrderHistory` now returns a Map rather than void
7. `CartProcessor.processRemoveItem` now returns a Map rather than void
8. `CartProcessor.processCheckoutAddresses` now returns a Map rather than void
9. `CartProcessor.processOrderComplete` now takes a Map as an argument rather than no arguments
10. `ShippingProcessor.processShipping` now returns a Map rather than void
11. `TaxProcessor.processTax` now returns a Map rather than void
12. `AdministratorGatewayDAO.processLogin` now returns a Map rather than void
13. `CustomerGatewayDAO.processRegister` now returns a Map rather than void
14. `OrderGatewayDAO.processOrderComplete` now takes an Order and a Map as arguments rather than just an Order

Version 1.0.8

Name Changes for .jar Files.

The following .jar files were renamed to eliminate the version numbers from the file names. This will allow for smoother upgrades of these files in future versions. Before upgrading to 1.0.8, please *delete* the following files from the application's `/WEB-INF/lib` directory. When you copy in the files for the upgrade, the new, renamed versions will take their place.

1. `jtds-1.0.2.jar`
2. `struts-menu-2.3.jar`
3. `mysql-connector-java-2.0.14-bin.jar`
4. `commons-codec-1.3.jar`

Appendix C. Migration Guide from 1.x to 2.x

Migration from 1.x to 2.x - Overview

Because of a number of architectural changes, the upgrade process from 1.x to 2.x does not follow the usual automated procedure. Integrating with the Hibernate persistence framework presented a number of opportunities to slim down the code base and the API, making it both more powerful and more streamlined. If we had attempted to create an automated upgrade process for 1.x installations we would have had to keep a large amount of code that was irrelevant for 2.x. Simplicity of design ruled the decision.

The basic process for migration is first to deploy the 2.x files; then to update the 1.x database, preserving your data; then to connect your 2.x installation to the database; and finally, to incorporate manually any 1.x customizations you've made into the 2.x installation.

In many cases upgrading to 2.x may not make sense. We have every intension to support 1.x as it's own code branch, and will continue to fix any bugs found in the future. (We do not expect to add any major new features to 1.x, however.) If you have made wide-ranging customizations to your store, or if your store would not be improved by the major new features of 2.x, you might save yourself a lot of hastle by *not* migrating.

Migration from 1.x to 2.x - Procedure

1. **Prepare for the Migration.**

    a. *Make Sure Migrating is Right for Your Store.* Is it worth migrating from 1.x to 2.x? The answer to that question depends on how much the new features of 2.x will improve your store, as well as to what extent you've customized your existing 1.x store. We intend to continue to support version 1.x, including fixing any new bugs found. (However, we do not expect to develop any additional features for 1.x.) The major new features of 2.x include integration with the Hibernate data persistance framework, inventory tracking, and discounting. For a complete list of changes, visit http://www.softslate.com/documentation/html/changeLog.html [http://www.softslate.com/documentation/html/changeLog.html] .

    b. *Set up a Staging or Test Environment.* We strongly recommend performing the migration on a staging or test server. Particluarly if you've made customizations to your 1.x installation, the migration process could take a while, and you don't want your store to be offline during the whole process.

    c. *Make a Backup of the SoftSlate Commerce Database.* Make a back up copy of the current 1.x application database. This is a very important step. If anything goes wrong you can restore the back up and investigate.

    ### Important

    Please don't skip this step! Making a backup copy of the database is a very important step, because it means you can always go back to the 1.x version of the application if anything goes wrong.

2. **Deploy SoftSlate Commerce 2.x.**

a. *Download SoftSlate Commerce 2.x.* Download the SoftSlate Commerce 2.x application from the URL given to you. If you don't know where to find the files, contact sales or support.

b. *Upload and Deploy SoftSlate Commerce 2.x.* Upload the application file for 2.x to your staging area and deploy its contents to your application server. Follow the installation steps outlined in the installation documentation to deploy the application.

Note

Do not run the installer tool or create an empty database. You'll be using the same database as your 1.x installation, after completing the next steps.

3. **Update the Database and Connect 2.x to it.**

   a. *Update the 1.x Database.* Next, update the database of your 1.x installation with all the new and modified database objects of 2.x. To do this, find the `/WEB-INF/classes/resources/upgrades` directory within the 2.x installation you just deployed. Against your 1.x database, run all of the SQL statements in all of the `.sql` files pertaining to your 1.x installation up to and including `upgrade_2.0.3_2.0.4.sql`.

      For example, if your current 1.x version is 1.0.9, you are upgrading to 2.0.4, and your database is MySQL, you would need to run the following files on your database, in this order:

      i. `upgrade_1.0.9_1.0.10.sql`

      ii. `upgrade_1.0.9_1.0.10_mysql.sql`

      iii. `upgrade_1.0.10_1.0.11.sql`

      iv. `upgrade_1.0.11_1.0.12.sql`

      v. `upgrade_1.0.12_1.0.13.sql`

      vi. `upgrade_1.0.13_1.0.14.sql`

      vii. `upgrade_1.99.99_2.0.1.sql`

      viii. `upgrade_1.99.99_2.0.1_mysql.sql`

      ix. `upgrade_2.0.1_2.0.2.sql`

      x. `upgrade_2.0.1_2.0.2_mysql.sql`

      xi. `upgrade_2.0.2_2.0.3.sql`

      xii. `upgrade_2.0.3_2.0.4.sql`

      To find out which 1.x version you are running, check the "databaseObjects" setting in the `npcSetting` table of the database, or go to the `Maintenance and Upgrades -> Upgrades` screen of the Administrator.

      As an example, for MySQL, you would run the following command to execute all the statements in one of the files:

      ```
      mysql -u <user> -p < upgrade_1.0.9_1.0.10.sql
      ```

   b. *Configure the 2.x Database Settings.* Next, copy the database settings from the 1.x installation

to the 2.x installation.

i. Find the `/WEB-INF/classes/appSettings.properties` file under the 1.x installation. This file contains your database connection settings.

ii. Find the `/WEB-INF/classes/hibernate.properties` file under the 2.x installation. This file tells Hibernate how to connect to the database in 2.x.

iii. If the "databaseType" setting is MySQL in 1.x, set the "hibernate.dialect" property to "org.hibernate.dialect.MySQLDialect" in the `hibernate.properties` file.

iv. If the "databaseType" setting is MSSQL in 1.x, set the "hibernate.dialect" property to "org.hibernate.dialect.SQLServerDialect" in the `hibernate.properties` file.

v. If you have set up a JNDI DataSource in 1.x, set the "hibernate.connection.datasource" property in the `hibernate.properties` file to the value of the "jndiContext" setting in 1.x, followed by a "/", followed by the value of the "dataSourceName" setting.

vi. Copy the value of the "databaseDriver" setting in 1.x to the "hibernate.connection.driver_class" property in the `hibernate.properties` file.

vii. Copy the value of the "databaseURL" setting in 1.x to the "hibernate.connection.url" property in the `hibernate.properties` file.

viii. Copy the value of the "databaseUserName" setting in 1.x to the "hibernate.connection.username" property in the `hibernate.properties` file.

ix. Copy the value of the "databasePassword" setting in 1.x to the "hibernate.connection.password" property in the `hibernate.properties` file.

x. If you've adjusted the connection pooling settings, you can make similar adjustments in the `hibernate.properties` file. Note that in 2.x the c3p0 connection pooling library is used, and you can find documentation for it by visiting the c3p0 web site [http://www.mchange.com/projects/c3p0/index.html].

xi. In the `appSettings.properties` file under the 1.x installation, find the value of the "twoWayKeyFile" setting. If you are using two-way encryption for passwords or payment information, you need to find this key file on the server and copy it to your 2.x installation, so the application can use it to decrypt existing information.

xii. Now find the `/WEB-INF/classes/appSettings.properties` file under the 2.x installation. Although `hibernate.properties` stores the database connection settings, this file still holds some important settings in 2.x.

xiii. After you copy the two-way key file, update the "twoWayKeyFile" setting in the `appSettings.properties` file under the 2.x installation to point to the file's location, so the application can find it.

xiv. Next, copy the value of the "databaseType" setting (MySQL or MSSQL) from the `appSettings.properties` file under the 1.x installation to the `appSettings.properties` under 2.x.

xv. Finally, change the "databaseTablePrefix" setting in the `appSettings.properties` file under the 2.x installation to "npc". (In 2.x, the database tables are prefixed with ssc by default instead of npc, but you may change the prefix by modifying this setting. Since you are using the same database as 1.x, the prefix should be set to npc.)

c. *Start and Test the 2.x Application.* You should now be able to start the 2.x application from

within your Java application server. For example, if you are using Tomcat use the Manager application to run the start command:

```
ht-
tp://localhost:8080/manager/start?path=/<path-to-2.x-installati
on>
```

Test the application to make sure it can connect to your database. You should see the application come up, but without any of the customizations you made to 1.x.

4. **Incorporate 1.x Customizations into the 2.x Installation.**

   For some suggestions and guidelines on incorporating your 1.x customizations into 2.x, please refer to the following section of this guide.

Incorporating 1.x Customizations into 2.x

Following is a list of the common ways to customize SoftSlate Commerce, and suggestions for how to proceed with incorporating each type of customization from 1.x to 2.x.

1. **Custom Java Classes.**

   A number of interface changes were made in the API from 1.x to 2.x. You will need to be aware of these changes, and may need to update your custom Java classes to conform to the new API. For a list of these interface changes, visit the following section of the Migration Guide.

   Another important change from 1.x to 2.x is that the custom implementations of the various interfaces are no longer stored as settings in the `npcSetting` database table. They are now stored in the `/WEB-INF/classes/appComponents.properties` file. When you are ready to test your custom classes, you must either update this file manually, or you may use the `Settings -> Components` screen to modify the file through the 2.x Administrator.

Note

To identify a custom implementation of a Java class in 2.x, you must either update the `/WEB-INF/classes/appComponents.properties` file manually, or you may use the `Settings -> Components` screen to modify the file through the 2.x Administrator.

2. **Custom SQL Queries.**

   If you created custom SQL queries for 1.x in one of more of the `sql-custom.properties` files found under the subdirectories of the `/WEB-INF/classes/resources` directory, you may create analogous queries in the `queries-custom.hbm.xml` files in the same directories in 2.x.

   Please be aware, however, that the `queries-custom.hbm.xml` files are used by Hibernate, and consist of queries written in the Hibernate Query Language (HQL) [http://www.hibernate.org/hib_docs/reference/en/html/queryhql.html]. A working knowledge of Hibernate and HQL is essential to writing custom queries.

Note

Because SoftSlate Commerce 2.x employs Hibernate, custom queries are written in the Hibernate Query Language (HQL), rather than SQL.

3. **Custom JSP Templates.**

   In a few cases, the default JSP templates have changed dramatically. Pay special note to the list of JSP template changes in the following section of the Migration Guide. If you created custom JSP template in the `/WEB-INF/layouts/custom` directory, they may need to be updated to interact correctly with the default 2.x templates. A good way to go about incorporating the customizations is to copy each custom JSP template over to the 2.x installation one at time, and testing each one for errors or other issues.

4. **Custom Struts Configurations.**

   Any custom Struts configurations you made in `/WEB-INF/conf/core/struts-config-custom.xml` should translate directly to 2.x. Simply copy the file over to 2.x, and test the custom form beans and actions to make sure there are no complications.

Note

Because of the interface changes from 1.x to 2.x, your custom Struts action classes may need to be modified to conform to the new API.

5. **Custom Tiles Definitions.**

   Custom Tiles definitions you made in `/WEB-INF/conf/core/tiles-defs-custom.xml`. should translate directly from 1.x to 2.x. Be sure to test them after copying the file from the 1.x to 2.x installation.

Interface Changes

Numerous interface changes were made to simplify the API and adapt it for use with the Hibernate persistance framework. Keep the following changes in mind as you incorporate any Java class customizations from 1.x into 2.x.

1. All objects implementing BusinessProcessor must now implement getSettings() and setSettings(Settings settings) rather than getSettingsBean() and setSettingsBean().
2. UpgradesProcessor, etc. that refer to SettingsBean must now refer to the Settings interface.
3. Custom objects subclassing BaseForm and BaseDynaForm must refer to getSettings() rather than getSettingsBean().
4. load*FromID() changed in all Processor classes to return a bean object rather than a Map.
5. Changed many property types in bean objects to wrapper classes for better integration with Hibernate. If the field was an int or double but nullable, it's now Integer or Double.
6. All the main Struts action forms in the Administrator for each of the administrator control screen have a new property: a String[] of "searchFields" containing the field names that are searchable.
7. Attribute property attributeOrder now an int.

8. Category getParentCollection is now a collection of parent Category objects, not the Integer Ids
9. Category must implement getParent, setParent, getProductCategories, setProductCategories
10. Product must implement getProductCategories, setProductCategories, getPrimaryCategory, setPrimaryCategory
11. processOrder() method of OrderGatewayDAO interface no longer part of the interface.
12. ProductGatewayDAO implementers now must implement Collection loadAttributeSKUs()
13. Category implementers must implement products property, for products with the category as a primary category.
14. Country must implement getStates(), and States must implement getCountry().
15. Administrator must implement decryptedPassword property.
16. Administrator now implements getRoleCodes() instead of getRoles()
17. AdministratorDAO no longer must implement updateAdministratorNoPassword().
18. AttributeDAO.loadAttributeOptions() removed.
19. ProductGatewayDAO.loadAllProductNamesCodesIDs()
20. Customer must implement decryptedPassword property.
21. CustomerDAO no longer must implement updateCustomerNoPassword()
22. Payment must implement decryptedNumber property.
23. Payment DAO no longer must implement updatePaymentNoNumber()
24. vvCustomerGatewayDAO.processLogin() now returns a Map not a Collection.
25. CustomerGatewayDAO.updateCustomer() now returns a Map not a Collection.
26. CustomerProcessor.loadCustomerLostPassword() now returns a Map not void
27. CustomerGatewayDAO.updateAddresses() now returns a Map not void.
28. CustomerGatewayDAO.processRegister() takes one argument - User.
29. CustomerProcessor no longer must implement the customer property.
30. ShipingProcessor.loadShippingOptions() can accept a Map as an argument.
31. ProductAttribute must implement new property productAttributeOrder

JSP Template Changes

In a few cases, the default JSP templates have changed dramatically. Because of this, we recommend installing 2.x from scratch and incorporating any custom JSP templates you created into the corresponding 2.x templates manually. As you do so, keep the following major JSP template changes in mind:

1. attributesAndOptions.jsp was replaced by various attributes templates, one for each attribute type.

If you've customized attributesAndOptions.jsp, integrate the customizations into the new templates.

2. In categories.jsp, the name of the property for the category's subcategories is now subcategoryCollection, not subcategories.

3. Placeholders for custom settings on the category and product pages have changed. For example, the category settings place holder changed from <bean:write name="categorySettings" property="value(color)"/> to <c:out value="${categorySettings.color.value}"/>

4. In paymentForms.jsp and paymentFormGuts.jsp, the reference to the activePaymentFormTemplates setting has changed to <bean:define id="activePaymentForms" name="appComponents" property="activePaymentFormTemplates"/>

Part II. Guide for Designers

Table of Contents

Chapter 10. Making Basic Customizations

Display Settings

As a designer, with SoftSlate Commerce, you have unsurpassed control over the look and feel of your store. By creating your own Custom Layout, you can manipulate literally every character of HTML the application produces. However, if you're not familiar with HTML or just want to get some basic design elements in place for your store, hundreds of thousands of design possibilities can be achieved by manipulating just a few settings on the `Settings -> Display` screen.

In particular, here are the display settings you're most likely to want to change:

- The "Store Logo" setting allows you to upload your own image as the logo that appears in the header of all pages. There are no hard and fast rules for the dimensions of your logo, but something in the neighborhood of 200 pixels by 100 pixels is recommended. If you do not upload a logo, your store's name will appear in the header instead. (The store name is defined on the `Settings -> Store` screen.)
- The "CSS Theme" setting allows you to select one of several pre-defined look-and-feel themes for your store. It's as simple as selecting the theme you want in the drop down menu. (If you know how to develop CSS stylesheets, you can add your own customizations and even create your own theme. See the next section for details.)
- The "Font Family" setting controls the font style of all the text displayed in your store. Switching this setting can make a big difference in the look and feel of the store.
- Finally, three separate settings control the colors used by the store: "Main Color", "Heading Color", and "Button Color". Setting these colors to match existing branding or experimenting with different options can go a long way toward getting the store to look the way you want it to.

Note

All of the settings defined under the "Settings" menu in the administrator are stored in the database in the `sscSetting` database table.

CSS Themes

If you're comfortable with CSS (Cascading Stylesheets), you'll want to know how to work with CSS Themes for your store. A CSS Theme in SoftSlate Commerce simply corresponds to a subdirectory under the `/css` directory. By default, SoftSlate Commerce uses the "base" CSS Theme. This simply means that it includes the `/css/base/base.css` stylesheet for all pages of the store. Each of the other themes that are distributed are extensions of the "base" theme. For example, if you select the "orangecrush" theme, the system will include the `/css/base/base.css` stylesheet first, followed by the `/css/orangecrush/orangecrush.css` stylesheet. The styles defined in `orangecrush.css` will extend and override the ones defined in `base.css`.

Note

> The `/WEB-INF/layouts/default-xhmtl/core/htmlHead.jsp` file is what includes each of the CSS stylesheets on all the store's pages. There's nothing wrong with creating a custom version of that file in the `/WEB-INF/layouts/default-xhmtl/custom/core` directory and modifying or hard coding which stylesheets are used, if you need to make customizations.

Selecting a CSS Theme is done in the Adminstrator by navigating to the `Settings -> Display` screen and selecting a theme for the "CSS Theme" setting. It is also possible to customize any of the built-in themes, or to create your own theme. To add or extend custom styles to any of the themes, simply do so in the `/css/custom/custom.css` file. That file will never be overwritten by installing a future upgrade, so your customizations will be safe. The system includes the `custom.css` stylesheet *after* both the base stylesheet and any theme you selected in the Administrator. Because of how CSS works, this means your customizations will override the styles defined before it.

Creating your own CSS theme altogether is simply a matter of creating a new stylesheet and placing it in a directory of the same name, under the `/css` directory. For example, to create a theme named "dynamite", you would create a subdirectory under `/css` named `dynamite`, and within that subdirectory, you'd place your stylesheet, giving the file the name `dynamite.css`. To make your theme active, you enter the name of your new theme in the "Other" field of the "CSS Theme" settings on the `Settings -> Display` screen.

It is hoped that 90% of the customizations you need to make to the design of your store can take place by customizing one of the built in CSS Themes. Sometimes, however, you just have to get into the templates and manipulate the HTML code that the store produces. Fortunately, as the next chapter explains, SoftSlate Commerce provides a easy way to customize its templates.

Chapter 11. Creating a Custom Layout

It's quite likely that before too long you will want to make more extensive changes to your store's look and feel than simply modifying the font styles and miscellaneous display settings. Fortunately SoftSlate Commerce gives you the ability to customize every piece of HTML produced by the application. It does so by allowing you to create your own custom JSP templates and to use them to replace the default templates that come with the application. Some familiarity with HTML is necessary for you to create your own templates. Familiarity with JSP and Struts tags is also a benefit, though not a necessity.

Setting Up Your Design Environment

Before making your first customizations, consider the design environment you'd like to use when creating your custom screens. There are many software products out there that provide features for editing HTML and JSP documents. Among the most popular are Macromedia's Dreamweaver [http://www.macromedia.com/software/dreamweaver] and the open-source editor jEdit [http://www.jedit.org/]. However, any text editor can be used to edit the JSP templates, including Microsoft's Notepad, or the open-source editor Emacs [http://www.gnu.org/software/emacs] .

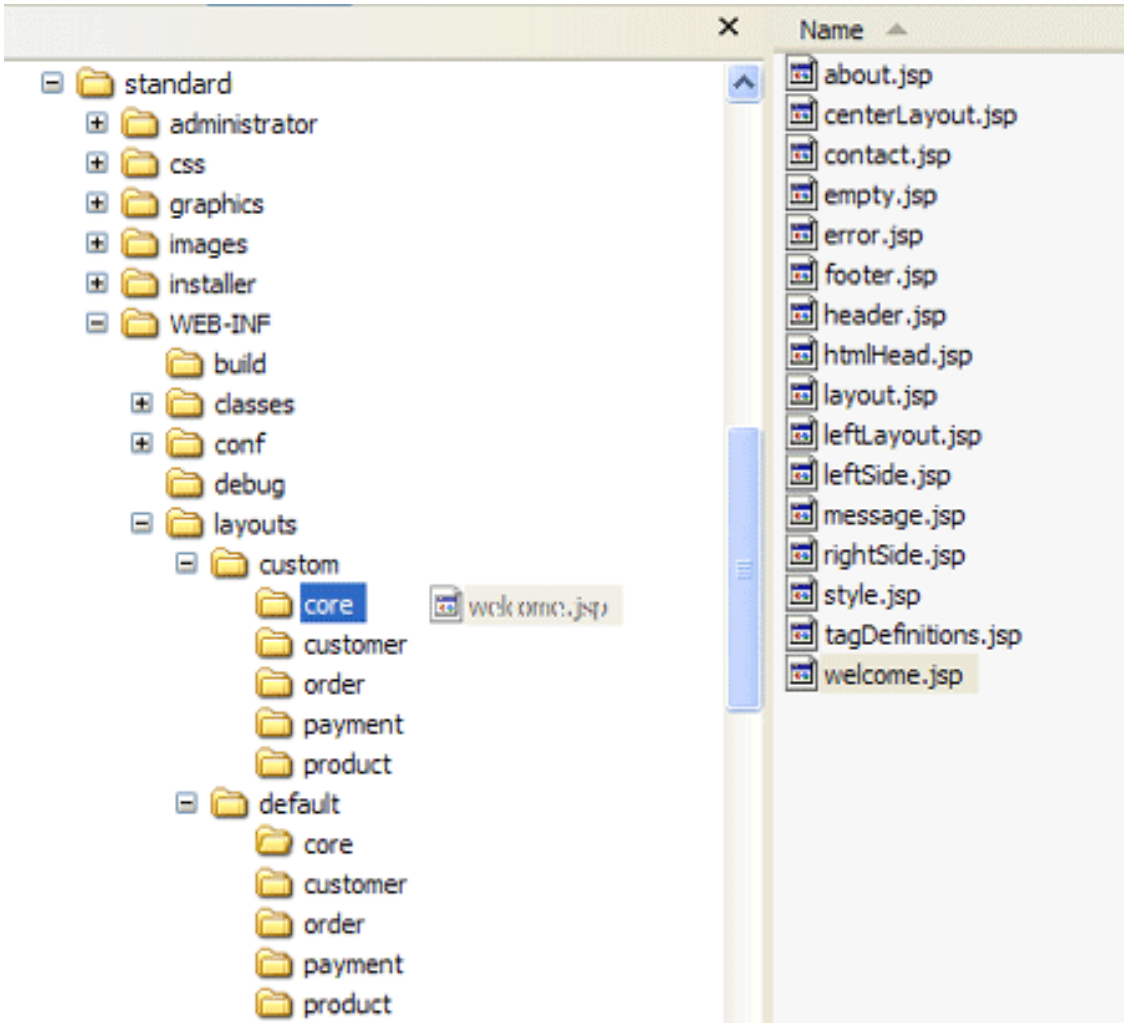
We strongly recommend setting up a separate installation of SoftSlate Commerce that you'll use only for development purposes. The SoftSlate Commerce license allows you to install the application on an unlimited number of development machines, so long as they are not accessible by the public through the Internet. It also allows you to install the application on a publicly-accessible *staging* server, which can be used to test your changes before going live.

Tip

An inexpensive way to set up a development installation of SoftSlate Commerce is to install Apache Tomcat [http://jakarta.apache.org/tomcat] and MySQL [http://www.mysql.com] on whatever machine you'll be using for development. Both Tomcat and MySQL are free, open-source products that work well with SoftSlate Commerce.

Custom and Default JSP Templates

The JSP templates that come with the application are all located in the `/WEB-INF/layouts/default` directory. Inside there, they are organized into subdirectories representing the major functional areas of your store: `core` , `product`, `order`, `customer`, and `payment` .



Copying a template from the `default` directory to the `custom` directory

Custom templates that you create will go in the `/WEB-INF/layouts/custom` directory, which contains the same five subdirectories. If a JSP template exists in the `custom` directory, the store will automatically detect and use it in place of the corresponding template under the `default` directory. This system allows you to create custom templates for just those areas of the store you need to modify, while other areas can be left untouched so that they use the default templates.

It's important not to modify the templates in the `/WEB-INF/layouts/default` directory. These files may be overridden when you install a future upgrade of SoftSlate Commerce. Instead, please place a copy of the template you wish to change into the corresponding subdirectory under `/WEB-INF/layouts/custom`. The store will automatically detect the new file and begin using it immediately. You can then feel free to make whatever changes you wish to the copied file.

Important

Do not modify the templates in the `/WEB-INF/layouts/default` directory. They may be overridden with future upgrades of SoftSlate Commerce. Instead, copy the template you wish to change into the corresponding subdirectory under `/WEB-INF/layouts/custom`.

Customizing Your First Screen

One of the first changes you're likely to make is to customize the Welcome Screen to display your own message to your visitors, featured products, or other information. Follow these steps to create a custom template for the Welcome Screen in place of the default template.

Example 11.1. Customizing the Welcome Screen

1. Refer to the Outline of the Default Welcome Screen. As you can see from the outline, the body of the Welcome Screen where the default heading and message appears, is produced by the `core/welcome.jsp` file. Find this file under the `/WEB-INF/layouts/default/core` directory.

2. You could make changes directly in this file, but you should *not*. The files inside the `/WEB-INF/layouts/default` directory may be overwritten when you install a future upgrade of SoftSlate Commerce. Instead, copy the `welcome.jsp` file into the `/WEB-INF/layouts/custom/core` directory.

3. Now you can safely make whatever changes you need to the copied `welcome.jsp` file under the `custom` directory. Open up the `welcome.jsp` you copied to the `custom` directory. It should look something like this:

```
<%@ include file="/WEB-INF/layouts/default/core/tagDefinitions.jsp" %>
<!-- Start welcome.jsp -->
<table width="100%" border="0" cellspacing="0" cellpadding="4">
        <tr>
                <td class="welcomeHeading">
                        <bean:message key="welcome.heading"/>
                </td>
        </tr>
        <tr>
                <td class="welcomeMessage">
                        <bean:message key="welcome.message"/>
                </td>
        </tr>
</table>
<!-- End welcome.jsp -->
```

4. Now replace the default welcome message with your own unique message:

```
<%@ include file="/WEB-INF/layouts/default/core/tagDefinitions.jsp" %>
<!-- Start welcome.jsp -->
<table width="100%" border="0" cellspacing="0" cellpadding="4">
        <tr>
                <td class="welcomeHeading">
                        <bean:message key="welcome.heading"/>
                </td>
        </tr>
        <tr>
                <td class="welcomeMessage">
                        Hello customers! So glad you decided to stop by!
                </td>
        </tr>
</table>
<!-- End welcome.jsp -->
```

5. Save the file. If necessary, upload the new file to the server you have running SoftSlate Commerce. Then navigate in your browser to the Welcome Screen [../../Welcome.do] (refresh the page if your browser is already there). You should see your changes on the screen.

Congratulations! You've just made your first customization to your store's screens.

Note

The <bean:message key="welcome.message"/> tag that was replaced in the above example is a tag that pulls in text from the `application.properties` files under the `/WEB-INF/classes/resources` directory, which are used to assist with internationalization and translations.

Customizing the Core Layout Files

Customizing a particular screen to add your own HTML, text, or graphics is one thing. But to implement a complete design, you will no doubt need to customize the core layout files that control the layout of each of the various screens in SoftSlate Commerce. These files provide the overall structure of each of the screen, while the other files in the application fill in the "guts" of the screens' content.

To customize one of the core layout files, use the same method as customizing any other JSP template: copy the default template from the `default` directory to the `custom` directory, and then make whatever changes you need to.

Note

You may find your design calls for using the Center Layout on all of the screens that by default use the store's Left Layout, or vice-versa. A simple technique for making this happen is to copy both `centerLayout.jsp` and `leftLayout.jsp` into the custom directory. Then, if you want to use the Center Layout in place of the Left Layout, you can simply replace the contents of `leftLayout.jsp` with the following:

```
<jsp:include page="centerLayout.jsp"/>
```

There are three layouts employed by the application: Center Layout, Left Layout, and Basic Layout. These layouts are produced by `core/centerLayout.jsp`, `core/leftLayout.jsp`, and `core/layout.jsp` respectively. Use the following table to find out which layout file corresponds to which screens.

Table 11.1. SoftSlate Commerce's Core Layouts and Their Screens

Layout	Template	Description	Screens
Center Layout	`core/centerLayout.jsp`	Features a column on both the left-hand and right-hand side of the screen. By default the left-hand side displays the store's category tree, a search box, and the user's cart. The right-hand column displays the store's Built-In Categories.	Welcome Screen
			About Screen
			Contact Screen
			Search Screen
			Cart Item Edit Screen
Left Layout	`core/leftLayout.jsp`	Features a column on the left-hand of the screen. By default the left-hand side displays the store's category tree, a search box, and the user's cart.	Category Screen
			Product Screen
			Product List Screen
			Search Results Screen
			Cart Screen
			Login Screen
			Register Screen
			Lost Password Screen
			Checkout Invite Login Screen
			Checkout Force Login Screen
			Checkout Invite Register Screen
Basic Layout	`core/layout.jsp`	Simply displays the screen's content in the middle of the page, between the header and footer. Used by default for the store's checkout screens and the customer account screens.	Error Screen
			Order Form Screen
			Account Addresses Screen
			Account Password Screen
			Account History Screen
			Account History Details Screen
			Checkout Addresses Screen
			Checkout Payment Screen
			Checkout Combo Screen
			Checkout ConfirmScreen
			Checkout Thank You Screen

Example 11.2. Customizing the Layout of the Checkout Screens

1. If you need to change the layout of the all screens in the checkout process, refer to the above table. As you can see the layout to customize is the Basic Layout, which is controlled by the `layout.jsp` file.

2. Make a copy of the `layout.jsp` file that appears in the `/WEB-INF/layouts/default/core` directory and place it in the `/WEB-INF/layouts/custom/core` directory. (As with the other JSP templates, you should place copies in the `custom` directory because the files in the `default` directory may be overwritten with a future upgrade.)

3. Now, feel free to make whatever changes you need to to the copied file. Upload the file into the SoftSlate Commerce application, and you're changes should take effect immediately, on all the screens that correspond to the layout file.

Customizing Screen Sections: Working with the Tiles Framework

SoftSlate Commerce uses the Tiles Framework [http://struts.apache.org/userGuide/dev_tiles.html] to organize its JSP templates. Throughout SoftSlate Commerce's JSP templates, you'll see tags like this, which invoke a Tiles definition:

```
<tiles:insert attribute="header"/>
```

If you aren't familiar with Tiles, you don't have to be to customize the templates. However, some understanding of Tiles and the Tiles definitions will help you find your way around and give you opportunities to customize your store in simpler ways.

"Tiles" can be thought of as different blocks on a page displayed in the browser. Which templates produce the different blocks are defined in Tiles definition files, which are xml files located in the `/WEB-INF/conf` folder of SoftSlate Commerce. For example, the Tiles definition for the Center Layout and for the Welcome Screen is found in `/WEB-INF/conf/core/tiles-defs.xml`. They look like this:

```
<definition name="core.baseCenterLayout"
        path="/WEB-INF/layouts/default/core/centerLayout.jsp"
        controllerUrl="/LayoutAction.do">
        <put name="htmlHead" value="/WEB-INF/layouts/default/core/htmlHead.jsp" />
        <put name="header" value="/WEB-INF/layouts/default/core/header.jsp" />
        <put name="error" value="/WEB-INF/layouts/default/core/error.jsp" />
        <put name="message" value="/WEB-INF/layouts/default/core/message.jsp" />
        <put name="leftSide" value="core.leftSide" />
        <put name="rightSide" value="core.rightSide" />
        <put name="subMenu" value="customer.accountMenu" />
        <put name="body" value="/WEB-INF/layouts/default/core/empty.jsp" />
        <put name="footer" value="/WEB-INF/layouts/default/core/footer.jsp" />
```

```
</definition>

<definition name="core.welcome" extends="core.baseCenterLayout">
        <put name="pageTitleKey" value="page.welcome"/>
        <put name="body" value="/WEB-INF/layouts/default/core/welcome.jsp" />
</definition>
```

These two definitions tell the Tiles Framework which JSP templates to use when displaying the Welcome Screen. They indicate that the `/WEB-INF/layouts/default/core/centerLayout.jsp` file is used to control the overall layout of the screen. Within that file, the definitions tell Tiles to include `/WEB-INF/layouts/default/core/header.jsp` where the `<tiles:insert attribute="header"/>` tag appears. Similarly, the `core.welcome` definition tells Tiles to include `/WEB-INF/layouts/default/core/welcome.jsp` when it comes across the `<tiles:insert attribute="body"/>` tag. In this way, the Welcome Screen is constructed by the Tiles Framework, using the JSP templates defined in the Tiles definition file.

Note

SoftSlate Commerce extends the Tiles framework to first look inside the `custom` directory each time a given template is called for, before using the template found in the `default` directory. This is how a definition that calls for `/WEB-INF/layouts/default/core/welcome.jsp` will include `/WEB-INF/layouts/custom/core/welcome.jsp` instead, if it exists.

There are some situations where the simplest way to customize a section of a screen is to change the Tiles definition for the screen rather than the JSP templates themselves. As an example, let's say you want your store's Welcome Screen to have a different header than the rest of your store's screens. By default, all of the store's screens include `/WEB-INF/layouts/default/core/header.jsp` to render the header. But you want the Welcome Screen to use a different template (maybe because you want to enlarge your logo, or include a different set of navigation links).

Example 11.3. Replacing the Welcome Screen's Header

1. The file `/WEB-INF/conf/core/tiles-defs-custom.xml` is included in SoftSlate Commerce to help with creating custom Tiles definitions. A custom definition will let us identify a special header template file for the Welcome Screen. Find the `/WEB-INF/conf/core/tiles-defs-custom.xml` file and open it in a text editor.

2. Add the following custom definition to `/WEB-INF/conf/core/tiles-defs-custom.xml`, inside the `<tiles-definitions>` tag:

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

<!DOCTYPE tiles-definitions PUBLIC
        "-//Apache Software Foundation//DTD Tiles Configuration 1.1//EN"
        "http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">


<!-- Tiles definitions for custom pages. -->

<tiles-definitions>
```

```
        <definition name="core.welcome" extends="core.baseCenterLayout">
                <put name="pageTitleKey" value="page.welcome"/>
            <put name="body" value="/WEB-INF/layouts/default/core/welcome.jsp"
            <put name="header"
                value="/WEB-INF/layouts/default/core/homeHeader.jsp" />
        </definition>


</tiles-definitions>
```

3. What this will do is override the default definition for the Welcome Screen. Where the default definition calls for `/WEB-INF/layouts/default/core/header.jsp` to be used for the header, this change will call up `/WEB-INF/layouts/default/core/homeHeader.jsp` instead, for just Welcome Screen. Save the `/WEB-INF/conf/core/tiles-defs-custom.xml` file and if necessary upload it to your installation of SoftSlate Commerce.

4. Of course, you now need to create the `homeHeader.jsp` file itself, with the changes you want to include on the Welcome Screen's header. Since this file is new and not part of the SoftSlate Commerce distribution, you can safely place it in the `default` directory. Doing so will let all of your custom layouts use the file as a default (which will come in handy if you end up creating Multiple Custom Layouts, described below). Create the `homeHeader.jsp` file now and if necessary upload it into the `/WEB-INF/layouts/default/core` directory.

5. Finally, whenver you make a change to a Tiles definition file, you must reload the SoftSlate Commerce application in your servlet container for the change to take effect.

6. After reloading the application, refresh the Welcome Screen in your browser. You should see the content of the `homeHeader.jsp` file appearing in place of the regular header.

Caution

The `/WEB-INF/conf/core/tiles-defs-custom.xml` file is configured to be read in by SoftSlate Commerce after all of the other Tiles definition files, so it can be used to override the existing definitions (as described above). Please use this technique to change a definition, and avoid modifying the other definition files. These other files may be modified or replaced when you install a future upgrade of SoftSlate Commerce.

Important

The three layout definitions, `core.baseLayout`, `core.baseLeftLayout`, and `core.baseCenterLayout` cannot be overridden using the `/WEB-INF/conf/core/tiles-defs-custom.xml` file. These definitions are extended by other definitions before the `/WEB-INF/conf/core/tiles-defs-custom.xml` file is read. The three definitions are exactly the same, so they can be interchanged. Consider using a JSP include command to include additional files from one of these core layout files.

Note

There is much more information about Tiles available in books and online. Refer to the Struts

Web site [http://struts.apache.org/userGuide/dev_tiles.html] for more details.

Chapter 12. Multiple Custom Layouts

We've seen examples of how to create a custom layout by copying the default JSP templates that come with the application into the `custom` directory and then making modifications. SoftSlate Commerce lets you take things one step further and create multiple layouts consisting of multiple sets of customized JSP templates.

In short, you can create as many directories under the `/WEB-INF/layouts` directory as you wish *next to* the `custom` and `default` directories, each one representing a completely separate look and feel for your store. Which layout a given visitor to your store sees can be controlled by a simple URL parameter (named *`layout`*). If the *`layout`* URL parameter matches the name of a directory under `/WEB-INF/layouts`, the store will use the templates in that directory, when they exist, to render the screens. (As with the `custom` layout, if a template does not exist, the store will use the corresponding template in the `default` directory.)

Adding an Additional Custom Layout

Example 12.1. Creating an Additional Custom Layout for Visitors Coming from an Affiliate

Let's say we want to create a special look and feel for people who visit the store from a link on a particular affiliate's site. (Perhaps we want to include the affiliate's logo in the store's header, or change the text or colors around a bit. Any customization on any or all screens is possible.)

1. The first thing to do is give a name to your new layout. For our purposes (and for lack of a more original name) let's call our new layout the "special" layout (a one-word term is best).

2. This name serves as both the value of the *`layout`* URL parameter for visitors coming to the site, and as the name of the directory under `/WEB-INF/layouts` that holds the new layout's customized templates. So, go ahead and create a subdirectory under `/WEB-INF/layouts` named `special`. While you're at it, within that directory create the same subdirectories found under the `custom` and `default` directories: `core` , `product`, `order`, `customer`, and `payment` .

3. Tell your affiliate to add the *`layout`* URL parameter, with a value of "special", to any links that take visitors from their site to your store. For example, the link to your Welcome Screen would look something lie this: `http://www.storesdomainname.com/softslate/Welcome.do?layout=special` . When SoftSlate Commerce sees that parameter on the URL, it knows to use the "special" layout throughout the user's session.

4. At this point you can follow the same techniques for creating a custom layout described in the Creating a Custom Layout chapter to create the new "special" layout. Copy files you want to customize from the `default` directory to the `special` directory. If the visitor is assigned to the "special" layout, the store will first look in the `special` directory to include each template. If it does not exist there, it will then include the corresponding template in the `default` directory.

Important

An important setting for controlling multiple custom layouts is the "Default Layout" setting in the `Settings -> Display` screen of the administrator. This setting tells SoftSlate Com-

merce which layout to give users by default, if the layout is not specified on the URL. For complete details, refer to the section on the Default Layout Setting

Chapter 13. More Customization Examples

SoftSlate Commerce's Built-In Categories

SoftSlate Commerce has four special Built-In Categories that you can use to display special products throughout the store. If you've created any of these categories in your store, SoftSlate Commerce will automatically load their products into memory, so you can use them to display special products on any screen. By default, the categories will be displayed on the right-hand side of the Welcome Screen.

The four Built-In Categories that SoftSlate Commerce looks for have the following category codes:

- _features
- _popular
- _favorites
- _new

These categories can be used, respectively, for featured products, most popular or bestselling products, favorites, and new products. (You can actually name the categories whatever you like, so long as the category code is one of the four codes.

As an example, let's add the "_features" category to the store, and then add some products to it.

Example 13.1. Adding a Built-In Category

1. To add a Built-In Category to your store, log into the administrator and navigate to the `Products -> Categories` screen. Click the "Add New Record" button on the right hand side of the screen to go to the add category form.
2. Add a category whose name is "Featured Products" and whose code is "_features". Make sure the code is entered correctly, including the underscore "_" at the beginning. If you enter it incorrectly, SoftSlate Commerce will not think it's one of the Built-In Categories and won't load its products into memory. For now leave the rest of the text fields empty. At the bottom of the form, for the "On Category Tree" field, select no, so the category does not appear on the store's category tree. Be sure to leave the "Active" field marked yes. Click "Add New Record" when you're done.
3. Now that you've added the category, let's assign some products to it. We'll assume you have already added one or more products, but if you haven't, refer to the Adding Your First Product section.
4. Navigate to `Products -> Categories`. You'll see the new "_features" category in the table in the center of the screen. Click the "Products" link next to it to view the products assigned and not assigned to the category. Click the "On" button next to "Edit Mode", check the checkboxes under the "Assigned" column next to the product you want to assign to the category, and click "Update/Delete" to save the change.

Chapter 14. SoftSlate Commerce's Screens

The following sections describe each of the screens in the customer interface for SoftSlate Commerce. When customizing or extending the application, it's essential to be familiar with the various screens and the roles they play. Use the following tables and sections as a reference when making changes.

Welcome Screen

By default the Welcome Screen uses `core/centerLayout.jsp` to control the overall layout of the page. Below is an outline of the Welcome Screen, with the JSP templates that it includes highlighted and labelled. You can use this outline to identify the areas of the Center Layout and of the Welcome Screen that you need to customize.

Table 14.1. Welcome Screen Reference

Path	Welcome.do [../../Welcome.do]
Required URL Parameters	None
Description	The opening screen of the store, and the target of the "Home" link in the default header.
Layout File	`core/centerLayout.jsp`
Key Included Files	`core/welcome.jsp`

Contact Screen

Table 14.2. Contact Screen Reference

Path	contactPage.do [../../contactPage.do]
Required URL Parameters	None
Description	A page intended to display the store's contact information. The target of the "Contact" link in the default header.
Layout File	`core/centerLayout.jsp`
Key Included Files	`core/contact.jsp`

About Screen

Table 14.3. About Screen Reference

Path	aboutPage.do [../../aboutPage.do]
Required URL Parameters	None
Description	A page intended to display general information

	about the store. The target of the "About" link in the default header.
Layout File	`core/centerLayout.jsp`
Key Included Files	`core/about.jsp`

Search Screen

Table 14.4. Search Screen Reference

Path	searchPage.do [../../searchPage.do]
Required URL Parameters	None
Description	Displays the search box form on its own page.
Layout File	`core/centerLayout.jsp`
Key Included Files	`product/search.jsp`

Cart Item Edit Screen

Table 14.5. Cart Item Edit Screen Reference

Path	CartItemEditForm.do
Required URL Parameters	*`orderItemID`*
Description	Displays information about one of the items in the user's cart, with buttons to edit or delete it.
Layout File	`core/centerLayout.jsp`
Key Included Files	`order/cartItemEdit.jsp`

Category Screen

By default the Category Screen uses `core/leftLayout.jsp` to control the overall layout of the page. Below is an outline of the Category Screen, with the JSP templates that it includes highlighted and labelled. You can use this outline to identify the areas of the Left Layout and of the Category Screen that you need to customize.

Table 14.6. Category Screen Reference

Path	Category.do
Required URL Parameters	*`code`*
Description	Displays all the information related to a given category, including a list of its subcategories, if any, and a list of its products.
Layout File	`core/leftLayout.jsp`

Key Included Files	`product/category.jsp`
	`product/subcategories.jsp`
	`product/productList.jsp`

Product Screen

Below is an outline of the Product Screen, with the JSP templates that it includes highlighted and labelled. You can use this outline to identify the areas of the Left Layout and of the Product Screen that you need to customize.

Table 14.7. Product Screen Reference

Path	Product.do
Required URL Parameters	*`code`*
Description	Displays all the information related to a given product. Provides an "Add To Cart" button to add the product to the cart. Also includes the product's attributes and options.
Layout File	`core/leftLayout.jsp`
Key Included Files	`product/product.jsp`
	`product/attributesAndOptions.jsp`

Product List Screen

Table 14.8. Product List Screen Reference

Path	ProductList.do [../../ProductList.do]
Required URL Parameters	None
Description	Displays a list of *all* the active products in the store. Useful for stores with a small number of products.
Layout File	`core/leftLayout.jsp`
Key Included Files	`product/productListLayout.jsp`
	`product/productList.jsp`
	`product/pagination.jsp`

Search Results Screen

Table 14.9. Search Results Screen Reference

Path	Search.do
Required URL Parameters	*`searchString`*
Description	Displays a list of the active products matching a

	search string provided by the user in the search box.
Layout File	`core/leftLayout.jsp`
Key Included Files	`product/searchResultsLayout.jsp`
	`product/productList.jsp`
	`product/pagination.jsp`

Cart Screen

Table 14.10. Cart Screen Reference

Path	Cart.do [../../Cart.do]
Required URL Parameters	None
Description	Displays the current user's cart and its contents.
Layout File	`core/leftLayout.jsp`
Key Included Files	`order/cart.jsp`

Account Login Screen

Table 14.11. Account Login Screen Reference

Path	AccountLogin.do [../../AccountLogin.do]
Required URL Parameters	None
Description	Displays a form for a customer to log in to his or her account, and links to create a new account or retrieve a lost password.
Layout File	`core/leftLayout.jsp`
Key Included Files	`customer/loginLayout.jsp`
	`customer/login.jsp`
	`customer/lostPasswordLink.jsp`
	`customer/registerLink.jsp`

Register Screen

Table 14.12. Register Screen Reference

Path	RegisterForm.do [../../RegisterForm.do]
Required URL Parameters	None
Description	Displays a form for a user to create a new customer account.
Layout File	`core/leftLayout.jsp`

Key Included Files	`customer/registerLayout.jsp`
	`customer/register.jsp`

Lost Password Screen

Table 14.13. Lost Password Screen Reference

Path	LostPasswordForm.do [../../LostPasswordForm.do]
Required URL Parameters	None
Description	Displays a form for a customer to have a lost password sent to him or her.
Layout File	`core/leftLayout.jsp`
Key Included Files	`customer/lostPassword.jsp`

Checkout Invite Login Screen

Table 14.14. Checkout Invite Login Screen Reference

Path	CheckoutInviteLoginForm.do [../../CheckoutInviteLoginForm.do]
Required URL Parameters	None, but user must have an appropriate session status to access the checkout screens.
Description	Displays a form for a customer to log in to his or her account, and links to create a new account, retrieve a lost password, or continue through checkout without an account.
Layout File	`core/leftLayout.jsp`
Key Included Files	`order/inviteLoginLayout.jsp`
	`customer/login.jsp`
	`customer/lostPasswordLink.jsp`
	`customer/registerLink.jsp`
	`order/declineLogin.jsp`

Checkout Force Login Screen

Table 14.15. Checkout Force Login Screen Reference

Path	CheckoutForceLoginForm.do [../../CheckoutForceLoginForm.do]
Required URL Parameters	None, but user must have an appropriate session status to access the checkout screens.
Description	Displays a form forcing the customer to log in to his or her account, or to create a new account be-

	fore proceeding through checkout. Also displays a link to retrieve a lost password.
Layout File	`core/leftLayout.jsp`
Key Included Files	`order/forceLoginLayout.jsp`
	`customer/login.jsp`
	`customer/lostPasswordLink.jsp`
	`customer/registerLink.jsp`

Checkout Invite Register Screen

Table 14.16. Checkout Invite Register Screen Reference

Path	CheckoutInviteRegisterForm.do [../../CheckoutInviteRegisterForm.do]
Required URL Parameters	None, but user must have an appropriate session status to access the checkout screens.
Description	Displays a form for a user to create a new customer account.
Layout File	`core/leftLayout.jsp`
Key Included Files	`order/inviteRegisterLayout.jsp`
	`customer/register.jsp`

Error Screen

Table 14.17. Error Screen Reference

Path	Error.do [../../Error.do]
Required URL Parameters	None
Description	Displays any unexpected errors generated by the application, including any exceptions thrown from the request processing.
Layout File	`core/layout.jsp`
Key Included Files	`core/error.jsp`

Order Form Screen

Table 14.18. Order Form Screen Reference

Path	OrderForm.do [../../OrderForm.do]
Required URL Parameters	None
Description	Displays a standalone order form where the user can select which products to buy and enter in ad-

	dress, shipping and payment information in one step.
Layout File	`core/layout.jsp`
Key Included Files	`order/orderFormLayout.jsp`
	`order/orderProductList.jsp`
	`order/billingFormGuts.jsp`
	`order/deliveryFormGuts.jsp`
	`order/shippingFormGuts.jsp`
	`order/paymentFormGuts.jsp`

Account Addresses Screen

Table 14.19. Account Addresses Screen Reference

Path	AccountAddresses.do [../../AccountAddresses.do]
Required URL Parameters	None, but customer must be logged in.
Description	Displays a form for the customer to enter or modify his or her billing and delivery addresses.
Layout File	`core/layout.jsp`
Key Included Files	`customer/addresses.jsp`
	`order/billingFormGuts.jsp`
	`order/deliveryFormGuts.jsp`

Account Password Screen

Table 14.20. Account Password Screen Reference

Path	AccountPassword.do [../../AccountPassword.do]
Required URL Parameters	None, but customer must be logged in.
Description	Displays a form for the customer to modify his or her user name and password.
Layout File	`core/layout.jsp`
Key Included Files	`customer/password.jsp`

Account History Screen

Table 14.21. Account History Screen Reference

Path	AccountHistory.do [../../AccountHistory.do]
Required URL Parameters	None, but customer must be logged in.
Description	Displays a list of the previous orders placed by the

	customer.
Layout File	`core/layout.jsp`
Key Included Files	`customer/historyLayout.jsp`
	`customer/historyList.jsp`

Account History Details Screen

Table 14.22. Account History Details Screen Reference

Path	OrderDetails.do
Required URL Parameters	*`orderNumber`*
Description	Displays the details of a particular order the customer has placed previously.
Layout File	`core/layout.jsp`
Key Included Files	`customer/historyDetailsLayout.jsp`
	`customer/orderDetails.jsp`

Checkout Addresses Screen

Table 14.23. Checkout Addresses Screen Reference

Path	CheckoutAddresses.do [../../CheckoutAddresses.do]
Required URL Parameters	None, but user must have an appropriate session status to access the checkout screens.
Description	Displays a form for the user to enter in billing and delivery address information, and to select a shipping option.
Layout File	`core/layout.jsp`
Key Included Files	`order/checkoutAddressesLayout.jsp`
	`order/checkoutBreadcrumbs.jsp`
	`order/billingFormGuts.jsp`
	`order/deliveryFormGuts.jsp`
	`order/shippingFormGuts.jsp`

Checkout Payment Screen

Table 14.24. Checkout Payment Screen Reference

Path	CheckoutPayment.do [../../CheckoutPayment.do]
Required URL Parameters	None, but user must have an appropriate session status to access the checkout screens.

Description	Displays a form for the user to enter in payment information.
Layout File	`core/layout.jsp`
Key Included Files	`order/paymentLayout.jsp`
	`order/checkoutBreadcrumbs.jsp`
	`order/reviewOrder.jsp`
	`order/paymentForms.jsp`

Checkout Combo Screen

Table 14.25. Checkout Combo Screen Reference

Path	CheckoutCombo.do [../../CheckoutCombo.do]
Required URL Parameters	None, but user must have an appropriate session status to access the checkout screens.
Description	Displays a combined form for the user to enter in billing and delivery address information, to select a shipping option, and to enter in payment information.
Layout File	`core/layout.jsp`
Key Included Files	`order/comboFormLayout.jsp`
	`order/checkoutBreadcrumbs.jsp`
	`order/billingFormGuts.jsp`
	`order/deliveryFormGuts.jsp`
	`order/shippingFormGuts.jsp`
	`order/paymentFormGuts.jsp`

Checkout Confirm Screen

Table 14.26. Checkout Confirm Screen Reference

Path	CheckoutConfirm.do [../../CheckoutConfirm.do]
Required URL Parameters	None, but user must have an appropriate session status to access the checkout screens.
Description	Displays a summary of the user's current order, and a button for the user to confirm and finalize it.
Layout File	`core/layout.jsp`
Key Included Files	`order/checkoutConfirmLayout.jsp`
	`order/checkoutBreadcrumbs.jsp`
	`order/reviewOrder.jsp`

Checkout Thank You Screen

Table 14.27. Checkout Thank You Screen Reference

Path	None - this screen is reached when the user has successfully finished the checkout process.
Required URL Parameters	User must have an appropriate session status to complete the checkout process.
Description	Displays an invoice of the order the user has just completed..
Layout File	`core/layout.jsp`
Key Included Files	`order/thankYou.jsp`
	`order/orderDetails.jsp`

Part III. Guide for Developers

Table of Contents

Chapter 15. Setting Up Your Development Environment

Overview

SoftSlate Commerce is designed to be completely agnostic in terms of what tools you decide to use for your development environment. The directory structure of the application matches the structure of any Java web application conforming to the latest servlet specification. This means that most Java IDEs supporting web development should recognize and support the application.

This said, we realize that in practice some Java IDEs have different requirements or expectations. Many IDEs are designed to be used for developing applications from scratch, rather than starting with an existing code base, as you will be with SoftSlate Commerce. For this reason, we won't attempt to provide instructions for using SoftSlate Commerce with all sorts of IDE's, whose features and requirements may change. Nor do we dictate the environment you must use. Instead we'll simply provide insight into the tools we use to develop and customize SoftSlate Commerce, in the hopes that it's helpful.

As of this writing the SoftSlate Commerce development team uses the following tools as a development environment:

- *Java SDK:* Sun JDK 1.5. http://java.sun.com/j2se/downloads.html [http://java.sun.com/j2se/downloads.html] .
- *Java Web application server:* Apache Tomcat. http://tomcat.apache.org [http://tomcat.apache.org] .
- *Integrated Development Environment (IDE):* Eclipse. www.eclipse.org [http://www.eclipse.org] .
- *Database servers:* MS SQL Server Developer Edition. http://tomcat.apache.org [http://tomcat.apache.org] and MySQL http://www.mysql.com/downloads [http://www.mysql.com/downloads].

Apache Tomcat

Apache Tomcat is the reference implementation of Sun's servlet specification. It is a free, open-source Java application server that can be used both for development and production purposes. It may be downloaded free of charge from http://tomcat.apache.org [http://tomcat.apache.org]. Follow the instructions on the Tomcat web site to install it on your platform. If you are developing on Windows, an installation program is available to help get it installed.

To get SoftSlate Commerce up and running under Tomcat, simply place the extracted .tar.gz or .zip file, or the unextracted .war file in the `webapps` directory under the Tomcat installation, and restart Tomcat. The application should then be available by browsing to `http://localhost:8080/<installation-directory>` . If necessary, you can refer to the installation instructions outlined in the chapter on Installing SoftSlate Commerce.

While developing any Java application that employs Hibernate, as SoftSlate Commerce does, a common problem you might experience is `java.lang.OutOfMemory` errors when "reloading" the application using Tomcat's manager application. Because of this issue we recommend you *restart* Tomcat after each code change to test it, rather than *reload* the application within Tomcat. This will prevent the `java.lang.OutOfMemory` errors from occuring and may save you a lot of grief. If you installed Apache Tomcat on Windows, you have the option of adding it as a service to the Windows Services

menu. This can make it more convenient to trigger a restart. Otherwise, Apache Tomcat comes with start/stop scripts in the `bin` directory that you can also use.

Note

If you plan to use the Eclipse as your IDE for the SoftSlate Commerce, we suggest pointing your Eclipse project to the same directory Tomcat uses for the application (under Tomcat's `webapps` directory). Since Eclipse compiles your Java code as you save each file, this prevents you from having to compile manually before testing your changes. A simple restart of Tomcat is all you need to do.

Eclipse

As of this writing among Java IDEs, Eclipse is the most popular and the only one currently gaining market share. It is a free, open-source application, and runs on Windows, Linux, Mac OS X, and other platforms. It provides great support for Java development, and has an extensive network of plug-in developers.

To use Eclipse with SoftSlate Commerce, follow these steps:

1. **Install SoftSlate Commerce.**

   If you haven't already, install SoftSlate Commerce on your development computer, following the steps outlined in the chapter on Installing SoftSlate Commerce. Note that the license allows you to install SoftSlate Commerce on any number of development machines that are inaccessible to the public.

2. **Download and Install Eclipse.**

   Visit www.eclipse.org [http://www.eclipse.org] to download Eclipse for your platform. It is free and runs on multiple operating systems. Follow the instructions on www.eclipse.org to install Eclipse.

3. **Create a New Java Project.**

   Once Eclipse is installed, create a new Java project pointing to the the directory where SoftSlate Commerce is installed on your local computer. Go to `File -> New -> Project`, select `Java Project` and click next, give the project any name you like, and then select "Create project from existing source" and find the SoftSlate Commerce installation directory.

4. **Configuration Settings.**

   Eclipse may automatically configure SoftSlate Commerce correctly when you first create the project. In case it does not, check the "Java Build Path" settings for the project to make sure they are correct:

   - Go to the `File -> Properties -> Java Build Path` screen.
   - Under the "Source" tab, add the `<installation-drectory>/WEB-INF/src` directory as the application's only source folder.
   - Under the same tab, make sure the "default output directory" is `<installation-drectory>/WEB-INF/classes`.
   - Next, under the "Libraries" tab, click the "Add JARs" button and check off *all* of the .jar files under the `WEB-INF/lib` and the `WEB-INF/build` directories.

When you're done configuring these settings, Eclipse should be able to successfully compile the application. You'll be off and running.

Note

If you plan to use the Tomcat as your application server, we suggest pointing your Eclipse project to the same directory Tomcat uses for the application (under Tomcat's `webapps` directory). Since Eclipse compiles your Java code as you save each file, this prevents you from having to compile manually before testing your changes. A simple restart of Tomcat is all you need to do.

Useful Eclipse Plugins

One of the best things about Eclipse is the vast number of plug-ins being developed for it. Unfortunately, the wide variety of plug-ins also represents a challenge, because it can be hard to find the right one for the job sometimes.

Following are some of the plug-ins we've found to be the most useful (as of the time of this writing). If you invest a little time up front installing and experimenting with these and other tools like them, chances are you'll save a lot more time in the long run by leveraging them to become more productive.

MyEclipse

This is a powerful (and very large) set of over 200 Eclipse plug-ins that assist you with Java development of all kinds. A MyEclipse license is available as a yearly subscription (the rates are reasonable), from http://www.myeclipseide.com [http://www.myeclipseide.com]. Among the most useful tools for SoftSlate Commerce development are:

- Support for editing JSP pages, including syntax highlighting, JSP tag content assist (by pressing CTRL-space), on-the-fly compilation (so you can see any errors before having to test them in a browser), and automatic formatting and indentation.
- A Struts configuration file editor that provides a "Design View", allowing you to view the Struts action mappings in a flow chart format.
- A database browser, which allows you to navigate the database and make manual updates to it through a visual GUI interface, all from within Eclipse.

As of this writing, it is a bit tricky to get MyEclipse set up with an existing application's code base, with all its capabilities enabled. (It is much easier to start a new project from scratch.) Following are our recommendations for getting your Eclipse project set up as a MyEclipse Struts project:

1. Right-click on your project in Eclipse, and select `MyEclipse -> Add Web project capabilities`.
2. In the subsequent wizard, uncheck the "Create web.xml" checkbox, and leave the "Add J2EE libs to build path" checkbox checked before continuing.
3. After the Web project capabilities have been added, optionally delete the `META-INF` directory MyEclipse created.

4. Next, add Struts capabilities to the project. But before doing so, make a back-up of the `WEB-INF/web.xml`. MyEclipse will overwrite that file when you add Struts capabilities and you'll need to restore it.

5. Right-click on your project in Eclipse, and select `MyEclipse -> Add Struts capabilities` .

6. In the subsequent wizard, select "Struts 1.2", enter `/core/struts-config.xml` for the Struts configuration file, and uncheck the install Struts JARs and TLDs options.

7. After the Struts capabilities have been added, restore the original `WEB-INF/web.xml` file that you had backed up.

8. Optionally delete the `com.yourcompany.struts` Java package created by the wizard.

9. We suggest turning *off* HTML, JavaScript, and XML validation. Do so from the `Window -> Preferences -> MyEclipse -> Validation` screen.

10. We have found that for MyEclipse to recognize the JSTL and Struts tags in JSP files, you must restart Eclipse after adding the Web project and Struts capabilities.

Note

To view any of the application's Struts configuration files visually using the MyEclipse Struts Editor, right-click on the configuration file, and select "MyEclipse Struts Editor" from the menu.

Hibernate Tools for Eclipse

Hibernate provides several Eclipse plug-ins as part of their Hibernate Tools project (http://www.hibernate.org/253.html [http://www.hibernate.org/253.html]). Among the most useful of these for SoftSlate Commerce development is the Hibernate Console. This is an Eclipse perspective that allows you to access your database through Hibernate. It is a great way to develop in the Hibernate Query Language (HQL), because it will translate your HQL to SQL, allowing you to test your queries against your database directly.

Struts Console

Struts Console is a popular program for validating and viewing an application's Struts configuration files. It provides a very basic (not a flow chart) visual representation of the file, which you can also use to edit it. A plug-in for Eclipse is available from http://www.jamesholmes.com/struts/console/ [http://www.jamesholmes.com/struts/console/].

Amateras EclipseHTML Plug-In

If you're looking for a simple tool to provide syntax highlighting and content assist for JSP editing, this is a lightweight plug-in that might serve your needs just fine. It is available free of charge from http://amateras.sourceforge.jp/cgi-bin/fswiki_en/wiki.cgi?page=EclipseHTMLEditor [http://amateras.sourceforge.jp/cgi-bin/fswiki_en/wiki.cgi?page=EclipseHTMLEditor] . Note that this plug-in has a couple of dependencies that must be installed beforehand.

After installing the Amateras plug-in, you might try the following to activate content assist automatically from within JSP tags. In Eclipse, navigate to `Window -> Preferences -> Amateras -> Code Assist` , and add a space character to the "Auto activation trigger" field. This will pull up the

content assist menu as soon as you hit a space character from within a JSP tag.

Databases

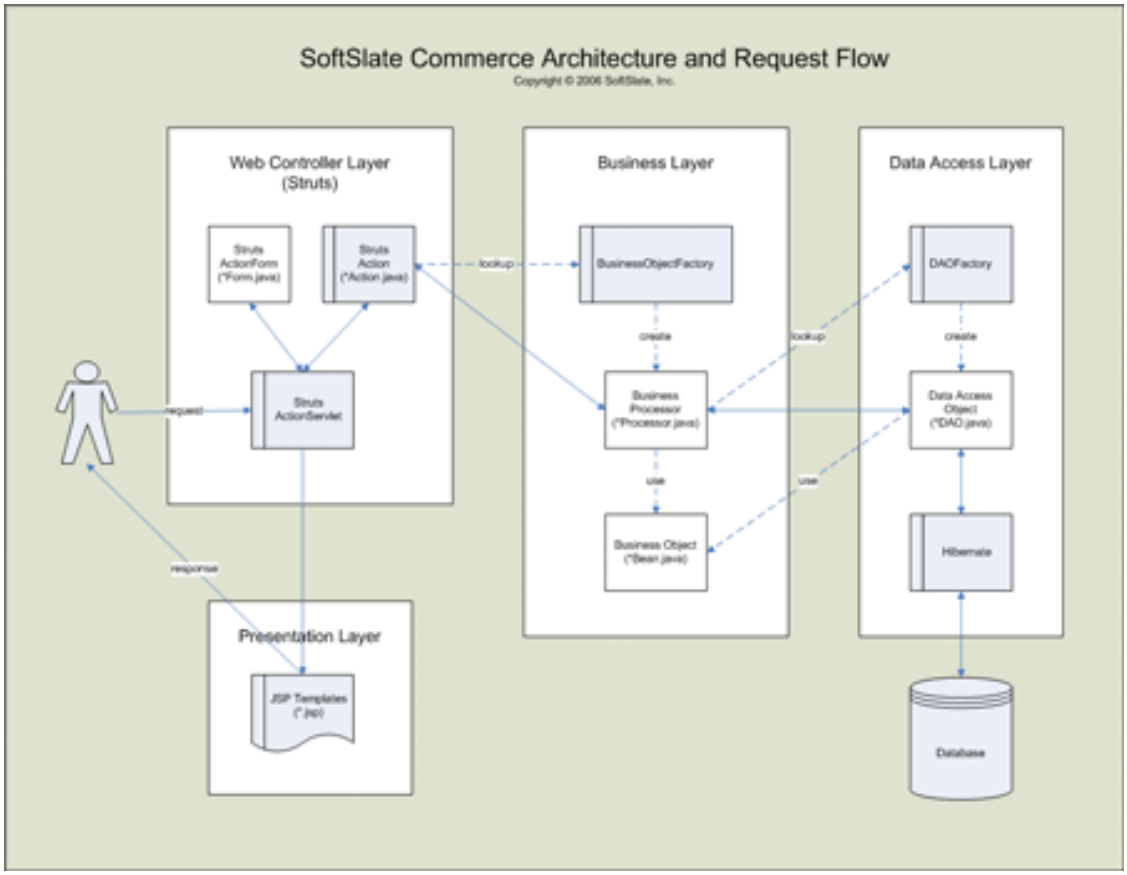
SoftSlate Commerce has been tested with Microsoft SQL Server 2000, MySQL, PostgreSQL, and Oracle 10g. We've found that the best database to use for development is largely a matter of the particular developer's preferences and experiences.

We often use MS SQL Server 2000 (http://www.microsoft.com/sql [http://www.microsoft.com/sql]), which of course is only an option if you're developing on Windows. It's Developer Edition is reasonably priced. MySQL http://www.mysql.com/downloads [http://www.mysql.com/downloads] is another favorite. It does not come with a tool to visually browse the database and make updates to it, but several options are available if that is a concern, including phpMyAdmin and the MyEclipse database browser.

Chapter 16. Overview for Developers

Application Architecture

SoftSlate Commerce uses a strict Model, View, Controller (MVC) architecture to organize its classes and files. The Model is further separated into two sections: classes devoted to interacting with the database (DAOs), and classes devoted to processing business logic. The result is four distinct layers to the application, illustrated by the following flow chart:



Application Architecture Flow Chart Larger Image [images/architecture.gif].

The Web Controller or Struts Layer

The Web Controller or Struts layer provides the mechanisms that control the routing of each web request through the system (the "C" in MVC). In SoftSlate Commerce there are three Struts modules, one for the main, customer interface, one for the administrator interface, and a small one for the installer tool. The customer interface and the administrator interface are further broken down into several units loosely organized by functional area, including "product", "customer", "order", "core", etc.

For the customer and administrator interfaces, the Web Controller or Struts layer consists of the following classes and files:

- The `struts-config*.xml` files under the `WEB-INF/conf` subdirectories (for the customer in-

terface).

- The `struts-config*.xml` files under the `WEB-INF/conf/administrator` subdirectories (for the administrator interface).
- The classes in the `com.softslate.commerce.customer` package, most of which are subclasses of `org.apache.struts.action.ActionForm` and `org.apache.struts.action.Action` (for the customer interface).
- The classes in the `com.softslate.commerce.administrator` package, most of which are subclasses of `org.apache.struts.action.ActionForm` and `org.apache.struts.action.Action` (for the administrator interface).

In short, the role of this layer is to take in each browser request, to validate the browser request, to invoke the classes in the Business layer responsible for processing the request, possibly to handle certain processing itself (such as recording information in the browser session), and finally to route the request to the appropriate Tiles definition in the Presentation layer.

Why separate the Web Controller into its own layer? Along with the Presentation layer, the Web Controller or Struts layer encapsulates all the processes that relate to SoftSlate Commerce being a Web application (as opposed to a GUI or some other application). For example, instances of `javax.servlet.http.HttpServletRequest` or `javax.servlet.http.HttpSession` are never referred to or used in the Business or Data Access layers. This separation makes it possible to use the same back-end processing with different front-end systems, such as legacy applications.

Note

Because of its use of the Struts web application framework, a familiarity with Struts is very useful when developing with SoftSlate Commerce. A great amount of information is available online and in books about Struts. For more, visit the Struts home page at http://struts.apache.org [http://struts.apache.org].

The Business Layer

The Business layer is responsible for processing the application's business logic and representing the application's state in the form of Java beans. For example, in the case of a request to add an item to the user's cart, the Business layer will initialize the user's cart if it hasn't been already, create new objects representing the items in the cart, and then communicate with the Data Access layer to store a representation of the cart in the database.

The Business layer consists of the following classes:

- The classes in the `com.softslate.commerce.businessobjects` package, which is broken down into several units loosely organized by functional area, including "product", "customer", "order", "core", etc.

The `com.softslate.commerce.businessobjects` package includes two types of classes. Processors (ie. those classes named `*Processor`) contain business processing logic, whereas Beans (ie. those classes named `*Bean`) store application state.

Each concrete processor class and bean class has a corresponding Java interface, which the rest of the application uses to instantiate and manipulate it. For example, the processor used to handle various requests related to a user's cart is named `com.softslate.commerce.businessobjects.order.BasicCartProcessor` .

However, throughout the rest of the application, instances of `BasicCartProcessor` are always referred to through its corresponding interface, `com.softslate.commerce.businessobjects.order.CartProcessor` . This practice provides a huge benefit in terms of customizing SoftSlate Commerce: through a simple configuration change, you can change the concrete class the application uses to something else. As long as it implements the original interface, the rest of the application can use it just as it did the original class. In your new custom class, however, you now have the ability to override any and every method to implement your own custom functionality. We'll explore the nuts and bolts of doing this in more detail later. The exact same feature is available for all the classes in the Data Access layer as well.

The Data Access Layer

The Data Access layer is responsible for writing and retrieving data to and from the database, and returning the results to the Business layer. For example, in the case of a request to add an item to the user's cart, the Business layer will invoke classes in the Data Access layer, which will do the work of inserting and updating the various tables in the database that store a representation of the user's cart. Starting with version 2.x, SoftSlate Commerce employs the Hibernate persistance framework to handle all of its database interaction.

The Data Access layer consists of the following classes and files:

- The classes in the `com.softslate.commerce.daos` package, which is broken down into several units loosely organized by functional area, including "product", "customer", "order", "core", etc.
- The `sql*.hbm.xml` files under the `WEB-INF/classes/resources` subdirectories, which contain the HQL (Hibernate Query Language) queries used by the application.
- The `*.hbm.xml` files in the `com.softslate.commerce.businessobjects` package, which contain the Hibernate mappings that map the business object Java beans to the database tables.

The Presentation Layer

After the Web Controller layer routes the request to the Business layer and retrieves the results of the processing, it then forwards the request to the Presentation layer, which is responsible for generating the HTML sent to the user's browser. The Presentation layer is composed of the following files:

- The `tiles-defs*.xml` files under the `WEB-INF/conf` subdirectories (for the customer interface).
- The `tiles-defs*.xml` files under the `WEB-INF/conf/administrator` subdirectories (for the administrator interface).
- The JSP templates in the `WEB-INF/layouts` directory (for the customer interface).
- The JSP templates in the `WEB-INF/templates/administrator` directory (for the administrator interface).
- The `application.properties*.xml` files under the `WEB-INF/classes/resources` subdirectories, which contain various text messages used in the Presentation layer.
- The `css/style.css` and `css/style-custom.css` files, which contain the CSS stylesheet definitions the application uses.
- The `images` directory, which contains some images referred to in the application's HTML output.

For example, in the case of a request to view a product detail page, the Web Controller layer will place the requested product information retrieved from the database in the request scope, (in the form of an instance of `com.softslate.commerce.businessobjects.product.Product`). It will then forward the request to the Presentation layer, to output the HTML containing the information to the user's browser.

SoftSlate Commerce employs the Tiles framework, which is distributed as part of Struts, to organize the JSP templates responsible for presentation. When the Web Controller layer executes a forward to the Presentation layer, it specifies one of the Tiles definitions defined in the `tiles-defs*.xml` files under the `WEB-INF/conf` subdirectories. The role of each Tiles definition is to describe which JSP templates should be used to output the response.

Note

There is much more information about Tiles available in books and online. Refer to the Struts Web site [http://struts.apache.org/userGuide/dev_tiles.html] for more details.

In the case of a request to view a product detail page, the Web Controller layer forwards to the `product.product` Tiles definition, which is defined in the `WEB-INF/conf/product/tiles-defs-product.xml` file. That file tells Tiles to use the `WEB-INF/layouts/default/product/product.jsp` JSP template to display the body of the product page.

Note

SoftSlate Commerce extends the Tiles framework to first look inside the `custom` directory each time a given template is called for, before using the template found in the `default` directory. This is how a definition that calls for `/WEB-INF/layouts/default/core/welcome.jsp` will include `/WEB-INF/layouts/custom/core/welcome.jsp` instead, if it exists.

Database Structure

Refer to the following database diagram to gain an understanding of the database structure used by SoftSlate Commerce. For more details about the attributes of each table's fields, refer to the Database Dictionary [../databaseDictionary.html] .

SoftSlate Commerce Database Schema. Larger Image [images/full2.0.gif].

Here is a list of some of the notable aspects of the database structure:

- The `sscSetting` table stores all of the application's settings. It is designed to store settings of various data types. For each record, the value of the `valueType` field identifies which field is used to store the setting's value. In the application, the values are derived and `sscSetting` is converted into a `java.util.HashMap` in which the `code` value is the map's key. This map is stored in the application scope.

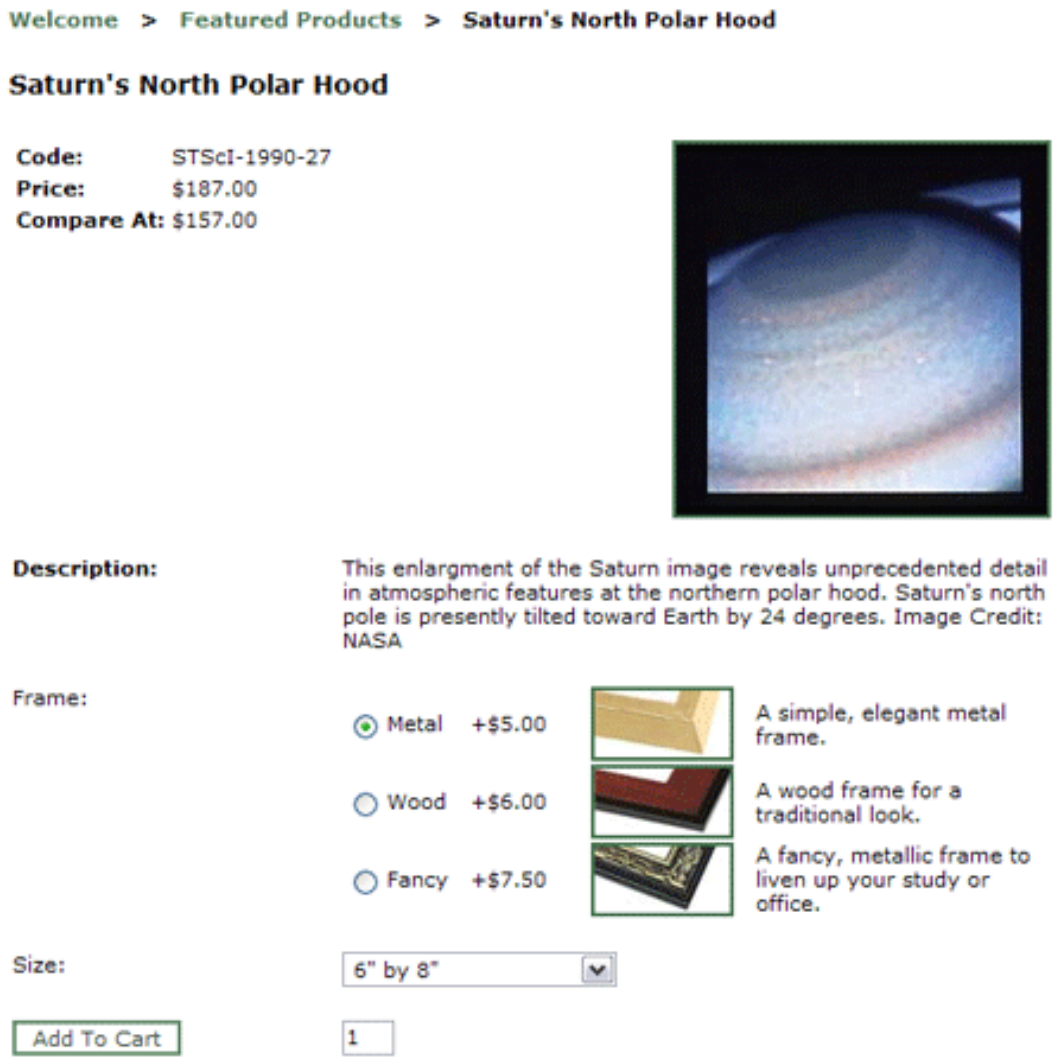
- Both complete and incomplete orders are stored side by side in the `sscOrder` table. Incomplete orders store a representation of the user's cart before completing checkout in the store. To test if a given order is complete, check that its `completed` field is not null. A tool exists in the Administrator (under the `Maintenance and Upgrade` menu), to clear out old incomplete orders from the database.

- The `sscOrder` and `sscCustomer` tables store the user's billing address, where as the `sscOrderDelivery` and `sscCustomerAddress` tables store delivery addresses. There may be many delivery addresses per order and per customer. However, the application's code does not currently support the notion of delivering a single order to multiple addresses, or storing multiple delivery addresses for each customer account.

Anatomy of a Request: Adding an Item to the Cart

There's nothing better than a detailed example to provide some insight into the concepts presented above. In the following example, we'll go step-by-step through a typical request sent to the SoftSlate Commerce application, touching on each of the configuration files, Java classes, and JSP templates used to process it. The example we'll use is a request from a user to add an item to the cart.

1. *The user submits the "Add to Cart" form from a product detail page.* As you can see, the form contains an input box for the quantity, as well as any number of form elements describing customer-selected "attributes" for the product (in this case, "Frame" and "Size" are the two attributes assigned to this product).



The Add to Cart form.

2. *The Web Controller or Struts layer identifies the action mapping corresponding to the URL of the request.* In this case, the URL is `/CartAdd.do`. The Struts `ActionServlet` looks up the corresponding action mapping and finds it in the `WEB-`

INF/conf/order/struts-config-order.xml configuration file:

```
<action
        path="/CartAdd"
        type="com.softslate.commerce.customer.order.CartAddAction"
        name="cartAddForm"
        validate="true"
        scope="request"
        input="/Product.do">
        <forward name="failure" path="/Product.do"/>
        <forward name="success" path="cart.full"/>
</action>
```

3. *The Web Controller or Struts layer identifies and invokes the Struts action form corresponding to the request.* The above action mapping refers to the "cartAddForm" form bean, which is defined at the top of the same configuration file, WEB-INF/conf/order/struts-config-order.xml:

```
<form-bean
        name="cartAddForm"
        type="com.softslate.commerce.customer.order.CartAddForm">
</form-bean>
```

Because the action mapping's validate attribute is set to true, Struts knows that the request must be validated. In Struts, this means that the validate method of the action form com.softslate.commerce.customer.order.CartAddForm must be invoked.

4. *The validate method of com.softslate.commerce.customer.order.CartAddForm validates the request's parameters.* In this case, CartAddForm makes sure a postitive integer is present for the quantity to be added, and that if any of the product's attributes are required, they are present in the request.

Note

More details about the add to cart validation is available in the API Documentation for CartAddForm [../api/com/softslate/commerce/customer/order/CartAddForm.html] .

If the result of the validation is one or more errors, the request gets forwarded to the value of the action mapping's input attribute. If there are no errors, Struts proceeds to invoke the execute method of the action class com.softslate.commerce.customer.order.CartAddAction . We'll assume a successful result without errors.

5. *The execute method of com.softslate.commerce.customer.order.CartAddAction is invoked to process the request.* In this case, CartAddAction starts by creating an instance of com.softslate.commerce.businessobjects.order.CartProcessor , invoking its processAddItems method:

```
// Process the items
CartProcessor cartProcessor = (CartProcessor) baseForm
                .getBusinessObjectFactory().createObject(
                                "cartProcessorImplementer");
Map results = cartProcessor.processAddItems(PropertyUtils
                .describe(baseForm));
```

At this point, control of the request is passed from the Web Controller or Struts layer, to the Business layer.

Note

Whenever an object in the Business layer is created, the `com.softslate.commerce.businessobjects.core.BusinessObjectFactory` class is used. This class looks at the values in the `WEB-INF/classes/appComponents.properties` file to determine which concrete class to instantiate. In this case by default it instantiates `com.softslate.commerce.businessobjects.order.BasicCartProcessor`, which is the application's default implementation of `com.softslate.commerce.businessobjects.order.CartProcessor`.

6. *Now in the Business layer, the `processAddItems` method of `com.softslate.commerce.businessobjects.order.BasicCartProcessor` is invoked.* This method handles the business logic involved with the request to add an item to the cart. Namely, it instantiates the user's cart if it doesn't already exist. A user's cart is represented as an instance of `com.softslate.commerce.businessobjects.order.Order`. It then parses the incoming parameters and creates an instance of `com.softslate.commerce.businessobjects.order.OrderItem` to represent the order item.

   After performing some validations against the database, processing inventory, and processing discounts, it finally hands things off to the Data Access layer to record the results of the request in the database:

```
OrderGatewayDAO orderGatewayDAO = (OrderGatewayDAO) getDaoFactory()
                .createDAO("orderGatewayDAOImplementer");
orderGatewayDAO
                .processOrderItems(getUser(), newOrderItems, results);
```

Note

Whenever an object in the Data Access layer is created, the `com.softslate.commerce.daos.core.DAOFactory` class is used. This class looks at the values in the `WEB-INF/classes/appComponents.properties` file to determine which concrete class to instantiate. In this case by default it instantiates `com.softslate.commerce.doas.order.OrderGatewayDAOHibernate`, which is the application's default implementation of

`com.softslate.commerce.doas.order.OrderGatewayDAO`.

7. *Now in the Data Access layer, the* `processOrderItems` *method of* `com.softslate.commerce.doas.order.OrderGatewayDAOHibernate` *is invoked.* This method synchronizes the state of the user's cart with the database. Namely, it will insert or update records into the `sscOrder` and related tables, storing the new order item in the `sscOrderItem` table.

   The DAO will use Hibernate mappings to synchronize the application's objects with the database. In this case, the Hibernate mapping file `WEB-INF/classes/com/softslate/commerce/businessobjects/order/Order.hbm.xml` and other mappings files next to it are used to synchronize the application's objects with the database tables.

   Some DAOs will use HQL (Hibernate Query Language) mappings to communicate with the database. In cases where data is being retrieved from the database, the DAO will employ the HQL queries in the `queries.hbm.xml` files in the `WEB-INF/classes/resources` subdirectories to query the database through Hibernate.

8. *Assuming a successful result, control passes back to* `com.softslate.commerce.customer.order.CartAddAction`. Assuming everything goes well, control will pass back to the Web Controller or Struts layer. In this case, `CartAddAction` will perform some additional processing of the request. In particular, depending on the application's inventory settings, the results of the processing by the Business layer might indicate that one or more messages be displayed to the user concerning inventory levels, or that one or more "low stock emails" be sent to the store's administrator.

   When it is finished processing these possibilities, the `execute` method of `CartAddAction` forwards the request to a Struts action forward:

```
return mapping.findForward(forward);
```

9. *The Web Controller or Struts layer looks up the action forward returned by the action class.* The Struts action forwards are defined back in the Struts action mapping for the request. In this case the following two action forwards can be returned:

```
<forward name="failure" path="/Product.do"/>
<forward name="success" path="cart.full"/>
```

   If it's a failure, control of the request is passed to the "/Product.do" URL. In other words, the user is sent back to the product page where any errors that occured are displayed.

   If on the other hand everything went well and the processing succeeded, control is passed to "cart.full", which is not a URL but rather a *Tiles definition*. Forwards that use this sort of dot notation identify Tiles definitions.

   It's at this point that control passes to the Presentation layer.

10. *The Presentation layer looks up the Tiles definition corresponding to the action forward.* In this case, it finds the "cart.full" Tiles definition in the `WEB-INF/conf/order/tiles-defs-order.xml` configuration file:

```
<definition name="cart.full" extends="core.baseLeftLayout">
        <put name="pageTitleKey" value="page.cart"/>
        <put name="subMenu" value="product.breadcrumbs" />
        <put name="body" value="cart.fullLayout" />
</definition>
```

Tiles definitions can both extend other definitions, and include other definitions under them. As you can see, "cart.full" extends the "core.baseLeftLayout" definition, which is defined in the `WEB-INF/conf/core/tiles-defs.xml` file:

```
<definition name="core.baseLeftLayout"
        path="/WEB-INF/layouts/default/core/leftLayout.jsp"
        controllerUrl="/LayoutAction.do">
        <put name="beforeHTML"
                value="/WEB-INF/layouts/default/core/empty.jsp" />
        <put name="htmlHead"
                value="/WEB-INF/layouts/default/core/htmlHead.jsp" />
        <put name="tracking"
                value="/WEB-INF/layouts/default/core/tracking.jsp" />
        <put name="header"
                value="/WEB-INF/layouts/default/core/header.jsp" />
        <put name="error"
                value="/WEB-INF/layouts/default/core/error.jsp" />
        <put name="message"
                value="/WEB-INF/layouts/default/core/message.jsp" />
        <put name="leftSide"
                value="core.leftSide" />
        <put name="rightSide"
                value="core.rightSide" />
        <put name="subMenu"
                value="/WEB-INF/layouts/default/core/empty.jsp" />
        <put name="body"
                value="/WEB-INF/layouts/default/core/empty.jsp" />
        <put name="footer"
                value="/WEB-INF/layouts/default/core/footer.jsp" />
        <put name="afterHTML"
                value="/WEB-INF/layouts/default/core/empty.jsp" />
</definition>
```

Here you can start to understand how Tiles organizes the JSP templates used to display the results of every request. The "cart.full" definition extends "core.baseLeftLayout", which identifies many of the JSP templates used to display the application's HTML. Note that "core.baseLeftLayout" indicates that a JSP file named `empty.jsp` should be use for the "body" of the page. Fortunately, "cart.full" overrides this attribute - otherwise, only an empty space would be displayed in the page's body.

"cart.full" identifies another Tiles definition should be used for the body of the page: "cart.fullLayout". This definition appears just below "cart.full" in the same `WEB-INF/conf/order/tiles-defs-order.xml` file:

```
<definition name="cart.fullLayout"
        path="/WEB-INF/layouts/default/order/cart.jsp">
        <put name="couponFormGuts"
                value="/WEB-INF/layouts/default/order/couponFormGuts.jsp" />
</definition>
```

With this definition we've now seen nearly all of the JSP templates used to generate the response. In particular, we can guess that `cart.jsp` is going to provide the guts of the screen for us.

11. *The Presentation layer forwards the request to JSP template correponding to the `path` attribute of the Tiles definition.* In this case, because "cart.full" extends "core.baseLeftLayout", the request is forwarded to `/WEB-INF/layouts/default/core/leftLayout.jsp` .

12. *The JSP templates are processed, including processing of the Tiles tags.*

    A peek at `leftLayout.jsp` will give you a good idea of how the various attributes of each Tiles definition are used. For example, this line is the point at which the value of the "body" attribute (in our case, `cart.jsp`) is included:

    ```
    <tiles:insert attribute="body"/>
    ```

    All of `leftLayout.jsp` is processed in this way, including each of the JSP templates referred to by their Tiles attributes. The result is an HTML page displaying the contents of the user's cart, including the new item that has just been added to it.

Note

SoftSlate Commerce extends the Tiles framework to first look inside the `custom` directory each time a given template is called for, before using the template found in the `default` directory. This is how a definition that calls for `/WEB-INF/layouts/default/core/welcome.jsp` will include `/WEB-INF/layouts/custom/core/welcome.jsp` instead, if it exists.

Chapter 17. Extending SoftSlate Commerce

7 Simple Rules for Making Customizations

For any customization you make to SoftSlate Commerce, at all times try to follow these rules. It will save you work and spare you from issues that may occur when upgrading to a future version.

1. **Custom Java Classes.**

   Override existing Java classes by subclassing the appropriate class from the `com.softslate.commerce` package with your own new class. Then, change the setting under the `Settings -> Components` screen of the Administrator or manually in the `/WEB-INF/classes/appComponents.properties` file to tell the application to use your new class instead of the default one. (Requires application reload.) Do not modify any of the existing classes as they may change with an upgrade.

   For more information, refer to SoftSlate Commerce's Java API documentation available at http://www.softslate.com .

   For information on extending the built-in business objects, refer to the section on Extending Built-In Business Objects .

2. **Custom HQL and SQL Queries.**

   Override existing Hibernate queries (HQL) or create new ones in the appropriate `queries-custom.hbm.xml` file found under the subdirectories of the `/WEB-INF/classes/resources` directory. (Requires application reload.) Queries defined in the `queries-custom.hbm.xml` files will override the queries defined in the regular `queries.hbm.xml` files. Do not modify any of the `queries.hbm.xml` files as they may change with an upgrade.

3. **Custom Struts Configurations.**

   Override existing Struts action mappings and form bean definitions, or create new ones, in `/WEB-INF/conf/core/struts-config-custom.xml` . (Requires application reload.) Since this is the last file read into memory, it will override the mappings and form beans defined in the other Struts XML files. Do not modify any of the other Struts XML files as they may change with an upgrade.

4. **Custom Tiles Definitions.**

   Override existing Tiles definitions or create new ones in `/WEB-INF/conf/core/tiles-defs-custom.xml` . (Requires application reload.) Since this is the last file read into memory, it will override the definitions defined in the other Tiles XML files. Do not modify any of the other Tiles definition XML files as they may change with an upgrade.

   For more information, refer to the section on Working with the Tiles Framework

5. **Custom JSP Templates.**

   Override existing JSP templates by copying them from the `/WEB-INF/layouts/default` directory over to the `/WEB-INF/layouts/custom` directory and modifying them there. Create any new JSP templates in the `custom` directory as well. Do not modify the templates in the `de-`

fault directory as they may change with an upgrade. (In a similar way, to customize templates used in the Administrator, copy templates from /WEB-INF/templates/administrator/default to /WEB-INF/templates/administrator/custom.)

For more information, refer to the section on Creating a Custom Layout.

6. **Custom Application Messages.**

   Override existing application messages or create new ones in the appropriate application-custom.properties file found under the subdirectories of the /WEB-INF/classes/resources directory. (Requires application reload.) Do not modify any of the other application*.properties files as they may change with an upgrade.

7. **Custom CSS Styles.**

   Override existing CSS styles or create new ones in /css//custom/custom.css. Do not modify any of the other files in /css as they may change with an upgrade.

   For more information, refer to the section on CSS Themes .

Extending SoftSlate Commerce's Business Objects

One of the most common needs in terms of customizing SoftSlate Commerce is to extend the application's business objects to include additional data beyond what's supported by default. For example, in the case of a bookstore, you may need to store additional fields like subtitle, author, and publisher with your products. In other cases, for customers, you may need to track how they first found out about your store or other idiosyncratic fields.

Fortunately, SoftSlate Commerce provides three ways to extend its built-in objects. As you will see, each of the methods has its advantages and disadvantages. We hope that the first two methods are fairly self-explanatory: they require minimal or no custom programming to use. For the last method, we'll go through a detailed example.

Using Built-In 'Extra' Fields

A number of the database tables feature 'Extra' fields named extra1, extra2, extra3, etc. These fields are intended to store any custom, generic data not supported by the built-in fields. To use the various 'Extra' fields, simply populate them in the database, or by using the Administrator as you add and edit your objects.

Supported Objects.

- sscCategory
- sscCustomer
- sscCustomerAddress
- sscDiscount
- sscOrder

- `sscOrderDelivery`
- `sscOrderDiscount`
- `sscOrderItem`
- `sscProduct`
- `sscSKU`

Advantages.

- The 'Extra' fields are placeholders meant to be used for any custom, generic data, so no database schema changes are necessary.
- The 'Extra' fields get copied from product tables into the correponding fields of the order tables (e.g. `sscProduct.extra1` -> `sscOrderItem.extra1`) with each order, so they can be used during order processing in addition to being displayed in the product catalog.
- The 'Extra' fields can be manipulated in the Administrator alongside the other fields.

Disadvantages.

- It may not be clear within programming code or in the Administrator what each of the different 'Extra' fields represents.
- The data types of the 'Extra' fields are each VARCHAR(255), so there is no flexibility in terms of the fields' data types.
- There are a limited number of 'Extra' fields in each table that supports them, so you may run out.

Storing Custom Data in Settings Tables

The two objects that most commonly need to be extended are categories and products. For these objects, you have the option to define custom settings, which are stored for each product and category respectively in the `sscProductSetting` and `sscCategorySetting` tables.

Supported Objects.

- `sscCategorySetting`
- `sscProductSetting`

Advantages.

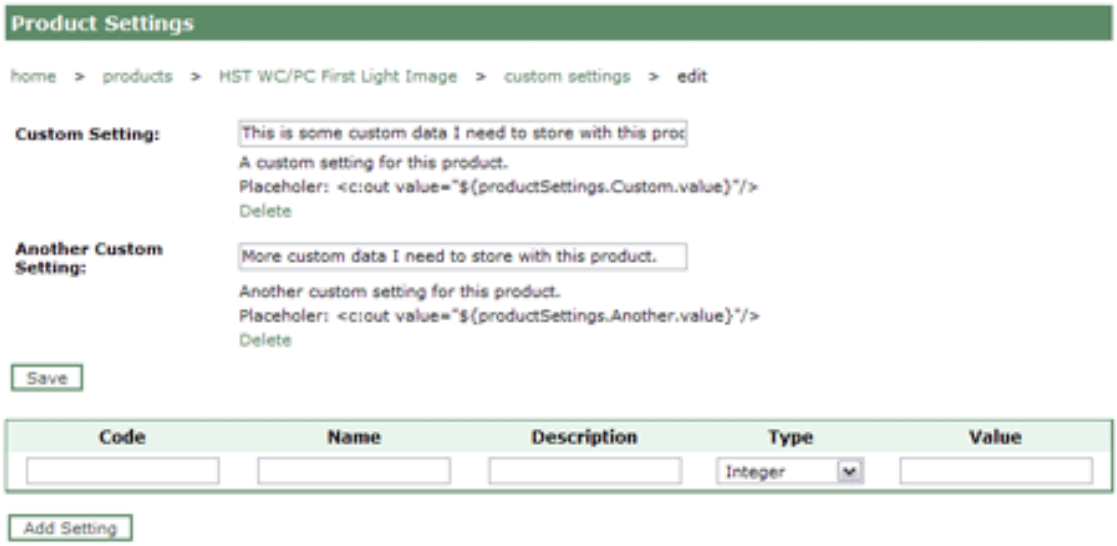
- Good for storing information that only needs to be used during the display of the product catalog. Product settings are loaded lazily on the product page, so no database programming is required to use them.
- The Administrator has an interface to create and modify any number of product settings for each product. (Navigate to `Products -> Products -> Details -> Custom Settings`)

- A similar interface, at `Products -> Categories -> Details -> Custom Settings`, is available for categories.

Disadvantages.

- Product settings are not copied into the order item objects as an order is placed, so they cannot easily be used during order processing.
- Even if multiple products or categories have the same set of settings, you must create them individually under each individual object.

To populate `sscProductSetting` and `sscCategorySetting` with custom data related to products and categories, the easiest way may be to use the Administrator's interface. For products, this interface is found by navigating to `Products -> Products -> Details -> Custom Settings`.



Product Settings Form.

On this screen, you find the following text input fields:

- *Code:* A unique string identifying this setting with the product. On the product page, use the code to display the value of the setting. For example, if you create a setting with a code of `Custom` you can display it by placing this tag in a custom `product.jsp` template: `<c:out value="${productSettings.Custom.value}"/>`
- *Name:* Used only in the Administrator, to explain to admin users what the setting is used for.
- *Description:* Also used only in the Administrator, to explain further to admin users what the setting represents.
- *Type:* Determines the data type of the setting. (Determines which field in `sscProductSetting` is used to store the value.)
- *Value:* The value of the setting for this product.

Adding Fields to the Database Tables

The method is both the most flexible and powerful, and also the one that takes the most work.

Supported Objects.

- All Objects

Advantages.

- Data can be stored as it ideally should - in the database table, with the correct data type. Queries can be designed to leverage the additional fields.
- If you add fields to one of the product tables, adding the same fields to the corresponding order table will allow the data to be copied over automatically with each order, and it can therefore be used during order processing logic. (E.g. `sscProduct.newField` -> `sscOrderItem.newField`.)

Disadvantages.

- Requires creating a number of customizations to support the additional fields, including a custom Java bean, a custom Hibernate mapping, and custom Administrator screens. (Much of the code is designed to support additional fields generically, however.)
- The table can get large if a large number of additional fields are required, possibly affecting performance or maintainability.

Let's go through an example of adding a field to a database table. In this example, we'll imagine we are running a bookstore, and that we've decided to add a field to `sscProduct` to store each book's author.

Example 17.1. Extending the Product Object by Adding a Field to `sscProduct`

1. Create a new class that extends the built in `com.softslate.commerce.businessobjects.product.ProductBean` class. To keep things clean, we strongly recommend you place all your custom classes in a separate Java package. We'll call our class `CustomProduct` and we'll put it in a new package named `com.softslate.commerce.demo` . Here's the source code for our new class:

```
package com.softslate.commerce.demo;

import com.softslate.commerce.businessobjects.product.ProductBean;

public class CustomProduct extends ProductBean {

        private static final long serialVersionUID = 6874908321131548936L;

        String author = null;

        public String getAuthor() {
                return author;
        }
```

```
        public void setAuthor(String author) {
                this.author = author;
        }

}
```

Note that we are extending the built-in `com.softslate.commerce.businessobjects.product.ProductBean` class. Strictly speaking, you don't have to extend the built-in class, but you must implement the `com.softslate.commerce.businessobjects.product.Product` interface.

2. Create the Hibernate mapping file for the new class. The Hibernate mapping file tells Hibernate how to map a class' properties to the columns of the database table it corresponds to. Make a copy of the `ProductBean.hbm.xml` file, place it next to the new `CustomProduct.class` file, rename it `CustomProduct.hbm.xml`, and replace the existing <subclass> tag with the following:

```
<subclass
        name="com.softslate.commerce.demo.CustomProduct">
        <property name="author" length="255"/>
</subclass>
```

If you are adding more than one property to your extended class, include all of them inside the `<subclass>` tag.

To ensure that your new class is used when Hibernate loads it lazily, add the `lazy="false"` attribute to the opening `<class>` tag:

```
<class
        name="com.softslate.commerce.businessobjects.product.Product"
        table="Product"
        lazy="false">
```

3. Now add the new column to the product database table:

```
ALTER TABLE sscProduct ADD author VARCHAR(255);
```

4. Update the product database table's `class` field to use the new custom class. This tells Hibernate to instantiate instances of the new class when loading data from the database:

```
UPDATE sscProduct SET class = 'com.softslate.commerce.demo.CustomProduct';
```

5. In the `/WEB-INF/classes/appComponents.properties` file, update the value of the productImplementer property with the fully-qualified classname of our new custom class:

```
productImplementer = com.softslate.commerce.demo.CustomProduct
```

   This tells the application to load the `CustomProduct.hbm.xml` file as part of the Hibernate initialization process.

6. To display our new author field, first put some data into the new column of the database:

```
UPDATE sscProduct SET author = 'Truman Capote' WHERE productID = 1;
```

7. Then, create a custom version of the `product.jsp` JSP template in the `/WEB-INF/layouts/custom/product` directory. Paste some JSP code into the custom template that displays the value of the new field if it is defined:

```
<c:if test="${!empty product.author}">
        <tr>
                <td class="productLabel" valign="top" nowrap="nowrap">
                        Author:
                </td>
                <td class="productData" valign="top">
                        <bean:write name="product" property="author"/>
                </td>
        </tr>
</c:if>
```

8. Finally, restart your application server or reload the application for your changes to take effect.

After extending the product object to add new properties, you may need to make them accessible in the Administrator so they can be viewed and edited there. The following example takes us through the steps required to add a new field to the Administrator interface.

Example 17.2. Adding a New Product Field to the Administrator Screens

1. Create a new class that extends the built in `com.softslate.commerce.administrator.product.ProductForm` class. This will handle the processing and display of the the control screen in the administrator. We'll call our class `CustomProductForm` and we'll put it in a package named `com.softslate.commerce.demo` . Here's the source code for our new `CustomProductForm` class:

```
package com.softslate.commerce.demo;

import java.util.Arrays;

import javax.servlet.http.HttpServletRequest;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionMessages;

import com.softslate.commerce.administrator.product.ProductForm;

/**
 * Adding author field
 *
 * @author David Tobey
 */
public class CustomProductForm extends ProductForm {

        private static final long serialVersionUID = 1571534730516598703L;

        static Log log = LogFactory.getLog(CustomProductForm.class);

        // Add the new field to the list of fields
        private String[] fields = { "productID", "code", "name", "keywords",
                        "shortDescription", "description", "isActive", "unitCos
                        "unitPrice", "altPrice", "weight", "header", "footer",
                        "smallImage", "mediumImage", "largeImage", "extra1", "e
                        "extra3", "extra4", "extra5", "primaryCategoryID",
                        "manufacturerID", "productOrder", "author" };

        // Add a human readable label for it
        private String[] fieldLables = { "Product ID", "Code", "Name", "Keyword
                        "Short Desc", "Description", "Active", "Cost", "Price",
                        "Alt Price", "Weight", "Header", "Footer", "Taxed", "Sm
                        "Medium Image", "Large Image", "Extra 1", "Extra 2", "E
                        "Extra 4", "Extra 5", "Pri Category", "Manufacturer", "
                        "Author" };

        // Should it be displayed on the control screen by default? Yes.
        private String[] displayFields = { "code", "name", "isActive", "unitPri
                        "productOrder", "author" };

        private String[] searchableFields = { "code", "name", "keywords",
                        "shortDescription", "extra1", "extra2", "extra3", "extr
                        "extra5", "author" };

        // Need a property for the control screen's edit mode - a String[]
        private String[] author = null;

        public String[] getAuthor() {
                return author;
        }

        public void setAuthor(String[] author) {
                this.author = author;
        }

        public String[] getDisplayFields() {
                return displayFields;
```

```
        }

        public void setDisplayFields(String[] displayFields) {
                this.displayFields = displayFields;
        }

        public String[] getFieldLables() {
                return fieldLables;
        }

        public void setFieldLables(String[] fieldLables) {
                this.fieldLables = fieldLables;
        }

        public String[] getFields() {
                return fields;
        }

        public void setFields(String[] fields) {
                this.fields = fields;
        }

        public String[] getSearchableFields() {
                return searchableFields;
        }

        public void setSearchableFields(String[] searchableFields) {
                this.searchableFields = searchableFields;
        }

        // Let's say we must have an author - so let's add it to the validation
        public ActionErrors validate(ActionMapping mapping,
                        HttpServletRequest request) {
                if (log.isDebugEnabled())
                        log.debug("Starting validation.");
                setProperties();
                // Run the original validations
                ActionErrors errors = super.validate(mapping, request);
                if (getProductID() != null) {
                        for (int i = 0; i < getProductID().length; i++) {
                                if (Arrays.asList(getDisplayFields()).contains(
                                        if (getAuthor().length < i + 1 || getAu
                                                || getAuthor()[i].equal
                                                errors.add(ActionMessages.GLOBA
                                                        new ActionMessa
                                        }
                                }
                        }
                }
                return errors;
        }
}
```

2. Similarly, create a new class that extends the built in `com.softslate.commerce.administrator.product.ProductAddEditForm` class. This will handle the processing and display of the the product detail screen in the administrator. We'll call our class `CustomProductAddEditForm` . Here's the source code for this class:

```
package com.softslate.commerce.demo;

import javax.servlet.http.HttpServletRequest;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionMessages;

import com.softslate.commerce.administrator.product.ProductAddEditForm;

/**
 * Adding author property
 *
 * @author David Tobey
 *
 */
public class CustomProductAddEditForm extends ProductAddEditForm {

        private static final long serialVersionUID = 4589930821507654940L;

        static Log log = LogFactory.getLog(CustomProductAddEditForm.class);

        private String author = null;

        public String getAuthor() {
                return author;
        }

        public void setAuthor(String author) {
                this.author = author;
        }

        public ActionErrors validate(ActionMapping mapping,
                        HttpServletRequest request) {
                if (log.isDebugEnabled())
                        log.debug("Starting validation.");
                initializeProperties(mapping, request);
                super.validate(mapping, request);
                if (getCode() == null || getCode().equals("")) {
                        getErrors().add(ActionMessages.GLOBAL_MESSAGE,
                                        new ActionMessage("errors.required", "A
                }
                return getErrors();
        }
}
```

3. Now let's tell Struts to use our new form classes instead of the defaults. Add the following Action Form definitions to the <form-beans> section of `/WEB-INF/conf/administrator/core/struts-config-custom.xml`:

```
                <!-- Adding an author field to the product admin screens  -->
                <form-bean
                        name="productAddEditForm"
                        type="com.softslate.commerce.demo.CustomProductAddEditF
                </form-bean>
                <form-bean
                        name="productForm"
```

```
                type="com.softslate.commerce.demo.CustomProductForm">
        </form-bean>
```

4. Next is to add the HTML form elements for the new field to our JSP templates. Let's start with the product detail screen. The JSP template responsible for it is `/WEB-INF/templates/administrator/default/product/productFormGuts.jsp`. We want to create a custom version of that file, so create a new file of the same name in the `custom` directory: `/WEB-INF/templates/administrator/custom/product/productFormGuts.jsp`

   In our custom version of `productFormGuts.jsp`, place the following content to display a text box for the new author field. Note we'll include the original JSP file to avoid duplication:

```
<%@ include file="/WEB-INF/layouts/default/core/tagDefinitions.jsp" %>

<!-- Start custom productFormGuts.jsp -->
<tr>
        <td class="genericLabel">Author:</td>
        <td>
                <html:text maxlength="100" name="productAddEditForm" property="
        </td>
</tr>
<tr>
        <td>

        </td>
        <td class="genericData">
                Enter the author of this book.
                <br />
                <br />
        </td>
</tr>

<jsp:include page="/WEB-INF/templates/administrator/default/product/productForm

<!-- End custom productFormGuts.jsp -->
```

5. Next is the control screen for products. The JSP template responsible for displaying the product fields on that screen is `/WEB-INF/templates/administrator/default/product/productFieldColumns.jsp`. We want to use the same file, but just add some new logic for the author field. So let's place a *copy* of `productFieldColumns.jsp` in the `custom` directory: `/WEB-INF/templates/administrator/custom/product/productFieldColumns.jsp`

   In our copy of `productFieldColumns.jsp`, *add* the following lines in the appropriate spots in that file to display the new author field, both when "edit mode" is on and when it is off. (Some analysis of the JSP tags is necessary here!)

   When edit mode is on:

```
<logic:equal name="field" value="author">
        <input type="text" size="15" maxlength="100"
                name="<bean:write name="field"/>"
                value="<bean:write name="item"
                property="<%= field %>"/>"
                class="genericGridData"/>
</logic:equal>
```

When edit mode is off:

```
<logic:equal name="field" value="partNumber">
        <bean:write name="item" property="<%= field %>"/>
</logic:equal>
```

6. Restart your application server or reload the application for your changes to take effect.

Appendix D. Included Libraries and Licenses

Table D.1. Included Libraries and Licenses

Library	JAR File	Version	License
Java Activation Framework	activation.jar	1.1	activation.license.txt
ANTLR	antlr.jar	2.7.6	antlr.license.txt
c3p0	c3p0.jar	0.9.1	c3p0.license.txt
cglib	cglib.jar	2.1.3	apache.license.txt
Apache Commons libraries	commons*.jar	Various	apache.license.txt
DHTML Color Picker	/administrator/colorpicker.html	1.0.3	http://creativecommons.org/licenses/by/2.5/
dom4j	dom4j.jar	1.6.1	dom4j.license.txt
Ehcache	ehcache.jar	1.2.3	apache.license.txt
Google Checkout	JavaCheckout*.jar		apache.license.txt
Hibernate	hibernate.jar	3.2.5	hibernate.license.txt
JDBC Interface Classes	jdbc-stdext.jar	2.0	jdbc-stdext.licence.txt
jTDS (SQL Server driver)	jtds.jar		jtds.licence.txt
Jakarta-Oro	jakarta-oro.jar		apache.license.txt
Java Standard Tag Library	jstl.jar	1.1.2	apache.license.txt
Java Transaction API	jta.jar	1.0.1B	jta.licence.txt
JavaMail	mail.jar	1.4	mail.license.txt
Jcaptcha	jcaptcha-all.jar	1.0 RC6	jcaptcha.license.txt
Log4J	log4j.jar		apache.license.txt
Lucene	lucene-core.jar	2.2.0	apache.license.txt
MySQL Connector	mysql-connector-		mysql.license.txt

Library	JAR File	Version	License
	java-bin.jar		
Opencsv	opencsv.jar	1.7	apache.license.txt
Oracle JDBC driver	ojdbc14.jar		oracle.license.txt
PostgreSQL JDBC driver	postgresql.jar		postgresql.license.txt
TagLib Standard	standard.jar	1.1.2	apache.license.txt
Struts	struts.jar	1.2.8	apache.license.txt
Struts-El	struts-el.jar	(Struts 1.2.8)	apache.license.txt
Struts Menu	struts-menu.jar	2.3	
PayPal Payflow Pro and PayFlow Link	Verisign.jar		
Xerces	resolver.jar, serializer.jar, xercesImpl.jar, xerces-Samples.jar, xml-apis.jar	2.9.1	apache.license.txt