Satchmo Documentation

Release 0.9.3-Dev

Chris Moffitt

September 19, 2014

Contents

Satchmo is an eCommerce framework created in Django which allows you to develop unique and robust online stores.

Overview

1.1 Satchmo Introduction

1.1.1 History

Like most Open Source projects, Satchmo was started to "scratch an itch." This particular itch was to create a framework for developing a python based shopping cart framework software using Django. After a little bit of discussion on the Django list, we created our own project in April 2006.

In August 2007, we released the first public release of Satchmo - version 0.5. We have had several releases since then and are continually improving the features and functions included in Satchmo.

1.1.2 Project Mission

Satchmo's mission is to use Django to create an open source framework for creating unique and robust online stores. To provide maximum flexibility, Satchmo is licensed under the BSD license.

1.1.3 Current Development Status

With the release of version 0.5, Satchmo entered beta status. The most recent release is 0.9.2 and includes many improvements and fixes over 0.9.1. All users are encouraged to use this latest version.

1.2 About this Project

This project was started by a group of individuals that were interested in using the Django framework to create a robust shopping cart solution. After some discussions, we have decided to focus on building a modular framework of shopping cart or eCommerce packages that can be easily put together to form a full store. Why did we use Django for this particular project?

First off, we wanted to use Python for the project. Many of us have experience with similar PHP based projects and we quickly realized that they tend to break down pretty quickly under the unwieldy syntax, lack of good OO support and hackish nature of many of the projects.

Once we decided on Python, there were many frameworks to evaluate. They all have tradeoffs but there are some things we really liked about Django:

- Robust, scalable system that has been used on major commercial sites

- Clean separation of program logic from presentation

- Nice abstraction of SQL (but with the capability to code SQL if required)

- Solid documentation

- URL mapping capabilities will be very useful for a shopping cart solution

- Out of the box admin capabilities making it easy to get up and running & provide enhanced security for multiple users in a store.

This list is not exhaustive and there could be endless debate and flamewars on the choices but we've all reached this decision independently and encourage you to investigate and make your own choice.

1.2.1 Why build a shopping cart framework?

Interestingly enough, there seems to be a fairly large number of "geeks" who are involved in creating web stores for their wives or significant others. For various reasons mentioned above, we independently started using Django. It just happens that there's the possibility for a lot of synergy here. Like most Open Source projects, we created this project because we thought there was an unmet need for ourselves as well as the larger audience.

Many of the current shopping carts out there were built a while ago and have not aged too gracefully. They also did not have the benefits of the Django framework. Looking at the Django framework, we realized that with just a little bit of additional work we could have a pretty powerful shopping cart system.

We initially thought about developing a full fledged shopping cart but realized that it is very difficult to build a one size fits all shopping cart package. Some people want a very simple solution and are confused by the multiple options and configurations available to them. Other people have the technical capabilities to build something very robust and tailored to their needs. In a similar way that Django "gets out of your way" and allows you to focus on the models and rules for your application, we want to build a framework that makes it trivial to setup a simple shop and relatively easy to modify it and grow as your business needs change.

1.2.2 What license are you using?

We have decided to use the BSD license for this project. You can learn more about it here .

1.3 Satchmo Features

1.3.1 Current Features

Satchmo strives to be extremely flexible.

- All display items are driven by templates using the powerful Django templating language

- All urls can be custom configured to your desired naming convention

- The checkout process can be tailored to your specific needs

Satchmo support many payment modules including:

- Authorize.net

- Trustcommerce

- Google checkout

- Cybersource

- Paypal
- Protx
- Sermepa
- Purchase orders

Satchmo has flexible shipping options and allows you to create your own. Satchmo includes:

- UPS integration
- Fedex integration
- USPS integration
- Canada Post integration
- Flat rate shipping
- Multi-tiered shipping based on quantity or price
- Per item shipping cost

Satchmo's has robust support for multiple product types including:

- Downloadable products
- Subscription products
- Custom configured products
- Product variants
- Gift certificates

All products offer you the opportunity to have:

- As many images per product as you would like
- Automatic thumbnail creation for the images
- Unlimited categories and sub categories
- Support for multiple pricing and discounting tiers based on volume
- Support for tiered/group pricing
- Inventory tracking including SKU's and preventing users from ordering out of stock items
- Meta data support for SEO
- Featured items
- Tax tables
- Related products
- Most popular products
- Arbitrary attributes
- Multiple translations per product
- Flexible variant creation (shirts with sizes and colors) including price changes for combinations
- Allow user to comment and rate products
- Comments support akismet spam tagging
- Support for brands

The customer model allows you to:

- Have multiple ship to and bill to addresses
- View order history
- Update account profile online
- Reset user passwords
- Require email verification for account creation

Satchmo supports discount codes which allow you to:

- Set amount of percentage discounts
- Limit the number of uses
- Allow free shipping
- Set start and end dates
- Limit to certain products

Satchmo includes extensive Internationalization and translation support:

- **Multiple translations are included:**
    - French
    - German
    - Italian
    - Spanish
    - Swedish
    - Bulgarian
    - Portuguese
    - Korean
    - Hebrew
    - Turkish
- Full country specific information
- Translation support for all products and categories
- Support for date and currency formating based on locale

Satchmo takes security seriously:

- **Django provides built in support to prevent many common attacks such as:**
    - SQL injection
    - Automatic HTML escaping to prevent cross-site scripting
    - Session forging/hijacking
- Satchmo encrypts all credit card information
- Satchmo allows you to choose if credit card data is stored
- Satchmo does not store ccv data in the database
- Fine grained ssl support for as many or as few urls as you need

Django is a proven scalable and robust system. Satchmo takes advantage of this by using:

- Django caching
- Opportunity to easily split out the tiers of the application (database, web, etc)
- A large suite of unit tests

In addition to these items, Satchmo provides:

- Generation of PDF invoices, packing slips and shipping labels
- Full store product searching
- Google analytic integration
- Google adwords support
- Google base feeds
- Newsletter support via mailman or custom database
- Recently viewed items
- Wishlists
- Ability to upsell products
- Define related products
- Multi-site capability

1.4 Requirements

Satchmo is based on the Django framework, therefore you do need a fully functioning Django instance to use Satchmo. The Django installation guide will step you through the process.

You must use Django 1.4.x. Satchmo is not compatible with Django 1.5.x at this time.

Satchmo requires Python 2.6 or later and a database supported by Django.

There is always a challenge in deciding how many dependencies to include in a project. With Satchmo, we strongly believe in avoiding "Not Invented Here" syndrome and using the power of the rich set of python tools available on the web to make Satchmo as flexible and powerful as possible.

There are a number of other Python packages that are required for usage of all the features in Satchmo.

- Satchmo's thumbnail capability is very robust and utilizes the following packages:
    - Pillow
    - Sorl Thumbnail

        New in version 0.9.3: - Use the latest version of Sorl Thumbnail

- In order to securely store sensitive information, you will need:
    - Python cryptography toolkit (Windows binary)
- Satchmo defaults to using HTML but can create PDF output for shipping and invoicing using:
    - ReportLab
    - Tiny RML2PDF (download link)

There are also a number of other Django packages (mentioned below) that you will need to install.

- Satchmo uses django-threaded-multihost internally for quick access to thread local variables (the current request, current user, etc.). Satchmo also uses this app for an optimization when running mutliple sites and store configurations with a single settings file (see *Basic Configuration* for more details):

  - Django Threaded Multihost

- For flexibility in defining template plugin points:

  - Django App Plugins

- For admin-configurable settings:

  - Django Livesettings

  New in version 0.9.1.

- For advanced caching settings:

  - Django Keyedcache

  New in version 0.9.1.

- For the account registration process, you will need:

  - Django Registration

A valid Django cache backend (file, memcached or DB) is required for the config settings.

- The following package is required to load the initial data and run the unit tests:

  - PyYaml

- Docutils is used for auto generating the admin documentation but is not required:

  - Docutils

- Sphinx is useful for auto generating the user documentation but is not required:

  - Sphinx

- If you wish to enable Akismet for spam prevention on product ratings, you will need to install:

  - Akismet

- Satchmo uses South for migrating database schema changes. It is not required to run the store but is very useful for migrating versions.

  - South

  New in version 0.9.1.

Detailed steps for installing these dependencies is included in the *Installation* section.

1.5 Directory Structure

Before proceeding too far with the Satchmo installation process, it is useful to get a basic understanding of the way Satchmo is laid out.

The base Satchmo directory should look like this:

```
|-- docs
`-- satchmo
    |-- apps
    |-- projects
    `-- static
```

The docs directory contains the text documents that can be used with Sphinx to create nicely formatted html or pdf documentation.

To understand the Satchmo template structure, please refer to *Template Customization*.

1.5.1 Satchmo Apps

The core Satchmo application is included in the apps directory and is laid out like this:

```
apps
|-- l10n
|-- payment
|-- product
|-- satchmo_ext
|-- satchmo_store
|-- satchmo_utils
|-- shipping
`-- tax
```

The directories are created in this fashion so that we can accomplish a couple of goals:

1. Decouple portions of Satchmo so that others may use and improve upon them

2. Bundle the required portions together so that installation is as simple as possible

3. Allow store owners flexibility in installing only the portions they need

4. Allow developers to extend Satchmo to provide the custom portions they require

Here's a brief description of the various applications.

l10n (required)   A collection of models and data used to Internationalize Satchmo. This data includes all of the country information as well as tools to present information correctly depending on the user's location.

payment (required)   The various payment modules, forms and views used to allow checking out of a store.

product (required)   The models and views used to store and present product information

satchmo_ext (optional)   A collection of Satchmo modules that provide optional features that may or may not be needed for your store. Additional information on these are described below.

satchmo_store (required)   The core models, views and urls used for a store. This application includes account information, contact information and the base tools for running a store.

satchmo_utils (required)   A collection of helper utilities used throughout Satchmo.

shipping (required)   The various modules used to determine shipping costs for orders.

tax (optional)   Modules for calculating tax based on various criteria.

Satchmo_ext

As described above, the satchmo_ext module includes many smaller applications that you may wish to include in your site.

brand   This application is useful if you have different brands of products that you want to use to categorize and display your products.

newsletter   Satchmo has two basic options for handling newsletter configurations. You can use a simple database list or interface with mailman. See *Newsletters*.

product_feeds Provides support for atom or csv feeds of the products in the store. For more info on using with Google base, see *Google Base*.

productratings Product ratings allows your store users to rate and review the products in the store.

recentlist Allows you to display recently viewed products on your site.

tieredpricing This application provides improved flexibility for charging different prices to different customers. It is useful for membership or other tiered price structures. See *Pricing Tiers*.

upsell Present options to customers to purchase other items based on their current items. See *Upsell*

wishlist The wishlist allows shoppers to add items from the store to a list that they might want to purchase from in the future.

1.5.2 Satchmo Projects

This directory contains an example project that illustrate how to layout Satchmo and integrate it with other Django applications. The installation process will discuss how to setup your individual application based on this sample. The simple project includes a sample database and can be run directly to provide a quick example of a Satchmo store.

Simple

This is a very basic example of a "simple" Satchmo store.

It should work right from checkout. Start it with "./manage.py runserver" and browse to http://localhost:8000/

Admin is:

Username: admin Password: simple

Skeleton

This directory includes the absolute basic files needed to get Satchmo working. It is not a standalone working version but is meant to be used as a template for your store. It is used by the clonesatchmo program to setup your store.

1.5.3 Satchmo Static

The static directory contains all of the css, javascript and image files used in Satchmo. This directory is meant to be copied to your local Satchmo store. Once your store is running this directory will contain your product images as well as custom css and javascript. The process for setting this up will be covered in the installation guide.

Installation

2.1 Quick Start

For the impatient, here is the quickest way to get Satchmo installed and running. If you plan to use Satchmo in a production environment, then it is important to understand the full *installation process*.

2.1.1 Install Base Requirements

Ensure that python 2.6 or greater is installed. Mercurial must also be installed.

Next, install python setuptools so that easy_install is available.

Install Pillow based on your OS.

Install pip:

```
easy_install pip
```

2.1.2 Install Satchmo and Dependencies

Execute these commands:

```
pip install -r http://bitbucket.org/chris1610/satchmo/raw/tip/scripts/requirements.txt
pip install -e hg+http://bitbucket.org/chris1610/satchmo/#egg=satchmo
```

Note: This will install the latest version of satchmo from tip.

2.1.3 Install the Satchmo Starter App

The clonesatchmo.py file should now be installed in your /bin directory. Use it to install the Satchmo directories and load the preliminary data:

```
cd /path/to/new/store
python clonesatchmo.py
```

Note: If you can not find clonesatchmo.py, it is included in the Satchmo distribution in /scripts/

2.1.4 Run the Development Server

Execute the development server command:

```
cd store
python manage.py runserver
```

2.1.5 Next Steps

You should review *Tutorial 1* to learn how to add Products to your store.

When your development is ready, you can "freeze" versions of packages, that you may want to install later on the production server:

```
pip freeze -r edited-requirements.txt > my-stable-reqirements.txt
```

2.2 Installation

This guide is the Satchmo installation process. It is meant to be a cookbook approach for most users. Advanced users may wish to modify this in order to integrate into their own projects. For the truly impatient, you may look at the *Quick Start* for the fastest way to get a store up and running.

This guide assumes you are working on a unix variant and that you are installing it somewhere into a directory you have write access to. In the below example, we use `/home/user/src`. You are expected to modify the path to fit your needs.

> **Warning:** You must have Django 1.4.x properly installed. Satchmo is not compatible with Django 1.5 at this time.

2.2.1 A Quick Note About Installing Dependencies

Python allows you to install applications in multiple ways: you can use the commands **easy_install** or **pip**; you may also manually install them by:

- linking the packages into your site-packages directory, or
- use .pth files to add each to your python path.

All of these will work fine with Satchmo, but in the interest of keeping this as straightforward as possible, we show how to install the packages with either:

- **pip**, or
- **python setup.py install** on a source code checkout (with mercurial)

Note: The provided egg and tar files by the various dependencies may not be the most current installation, so you should ensure that the version recommended by satchmo (see *Requirements*) is available via *easy_install* before proceeding.

2.2.2 Installing Dependencies

1. Install setuputils by following the instructions on the easy_install page. After installation, you should be able to run **easy_install** directly (assuming the install directory is in you $PATH).

2. Install required dependencies (this may vary based on your OS of choice):

   ```
   pip install pycrypto
   pip install pyyaml
   pip install pillow
   ```

3. Install django-threaded-multihost:

   ```
   pip install django-threaded-multihost
   ```

4. Install django-app-plugins:

   ```
   pip install django-app-plugins
   ```

5. Install sorl-thumbnail:

   ```
   pip install sorl-thumbnail
   ```

6. Install livesettings:

   ```
   pip install django-livesettings
   ```

7. Install keyedcache:

   ```
   pip install django-keyedcache
   ```

   New in version 0.9.1.

8. Satchmo has two types of documentation: Sphinx and docutils. Sphinx is used to generate this document, while docutils are useful for the auto-generated admin documentation.

   You may choose to install these dependencies by running:

   ```
   pip install sphinx
   pip install docutils
   ```

9. Satchmo also uses South for database migrations. You may also install it:

   ```
   pip install South
   ```

   See also:

   Satchmo Migrations and Upgrades for more information on how migrations in Satchmo should be applied.

   New in version 0.9.1.

10. If you use the Fedex shipping module, you will need to install 2 additional modules:

    ```
    pip install fedex
    pip install suds
    ```

11. Install django-registration:

    ```
    pip install django-registration
    ```

2.2.3 Installing Satchmo into your path

1. Checkout the latest Satchmo release into `/home/user/src`:

```
hg clone http://bitbucket.org/chris1610/satchmo/
```

Note: If you are a bitbucket user, you may see a slightly different url than described above. You may use the generic url or one that is specific to your username. For the purposes of an install, either will work.

2. Install satchmo onto your system:

```
cd /home/user/src/satchmo-trunk
sudo python setup.py install
```

Note: An alternative to running the install is ensuring that `/path/to/satchmo/apps` is on your python path. You may do this by placing a symbolic link to the source, adding a .pth file that points to your `/satchmo/apps` location or modifying your PYTHONPATH environment variable.

3. Once the above step is completed, you should be able to import both django and satchmo:

```
$ python
Python 2.5.2 (r252:60911, Mar 12 2008, 13:39:09)
[GCC 4.2.3 (Ubuntu 4.2.3-2ubuntu4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> import satchmo_store
>>> satchmo_store.get_version()
'0.9-pre hg-YYYY:ZZZZZZZZZZ'
```

Warning: Do not attempt to progress any further on the install until the above imports work.

2.2.4 Build Your Store With clonesatchmo

In order to quickly get you up and running, satchmo includes a helper script that will get a new sample store up and running quickly.

1. Build a starter store using **clonesatchmo**:

```
python /home/user/src/satchmo-trunk/scripts/clonesatchmo.py
```

The clonesatchmo program will do everything described in *Settings*. At a high level it:

- Creates a directory for your store (defaults to simplestore)
- Also creates a localsite directory for your customizations (defaults to localsite)
- Copies a valid settings.py and local_settings.py file
- Copies your static directory in place
- Copies a valid urls.py file
- Synchs your database (using sqlite)
- Loads l10n data
- Loads a sample store

Using this process is the recommended way to get your store up and running. Once you are comfortable with this store, you can dive into modifying your *Settings* file or making other changes.

2.2.5 View the Demo Store

After you have completed your initial install, you can check out your demo store using the commands below.

1. Start up the sample webserver to see your store:

   ```
   python manage.py runserver
   ```

2. In order to see your sample store, point your browser to:

   ```
   http://127.0.0.1:8000/shop or http://127.0.0.1/
   ```

3. If you want to see the admin interface, point your browser to:

   ```
   http://127.0.0.1:8000/admin
   ```

4. Many configuration and customization settings are accessed through the url:

   ```
   http://127.0.0.1:8000/settings
   ```

5. Additional detailed documentation can be found here:

   ```
   http://127.0.0.1:8000/admin/doc
   ```

Note: The above urls will be dependent on your Django setup. If you're running the webserver on the same machine you're developing on, the above urls should work. If not, use the appropriate url.

2.2.6 Troubleshooting

If after following these steps, you have errors or can not get the store to work, satchmo includes a custom command to check your system's configuration. To check your system out:

```
python manage.py satchmo_check
Checking your satchmo configuration.
Your configuration has no errors.
```

2.2.7 Additional Notes

Satchmo also includes a full set of unit tests. After you get your system installed, you can run the unit tests with this command:

```
python manage.py test
```

2.2.8 Useful Links

Django installation guide

2.3 Settings

The recommended way to get a working settings file and configure your first store is to use the *clonesatchmo* program.
However, if you would like to customize the settings by hand or dive into the details of how Satchmo's settings are
used, this portion of the documentation will be useful.

2.3.1 Customizing the settings

Once Satchmo is installed on your PYTHONPATH, you will need to create a new project or integrate Satchmo into an
existing project. Before proceeding with the next steps, please familiarize yourself with the sample projects, *see here
for more*.

Each of these projects show examples of how to configure Satchmo and integrate it with other Django applications.
Once you are familiar with these examples, you can configure your project based on the notes below.

Additionally, there is a streamlined satchmo directory structure in the skel directory. You may use this as the basis for
your store.

You need to customize the settings.py file in mystore to include the relevant satchmo information. A sample file called
`settings.py` is available in the projects/base directory to act as a template. You may use this file as a template for
`settings.py` or use the notes below to configure your existing one.

Please remember to ensure that your Django database connections and settings are working properly before trying to
add any pieces of satchmo.

1. Ensure that `/home/user/src/mystore/settings.py` has the following satchmo-specific configura-
   tions (in addition to the defaults and your other app needs):

```python
import os
DIRNAME = os.path.abspath(os.path.dirname(__file__))
LOCAL_DEV = True

MEDIA_ROOT = os.path.join(DIRNAME, 'static/')
MEDIA_URL = '/static/'

MIDDLEWARE_CLASSES = (
    "django.middleware.common.CommonMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.locale.LocaleMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.middleware.doc.XViewMiddleware",
    "threaded_multihost.middleware.ThreadLocalMiddleware",
    "satchmo_store.shop.SSLMiddleware.SSLRedirect",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware")

TEMPLATE_DIRS = (os.path.join(DIRNAME, "templates"))
TEMPLATE_CONTEXT_PROCESSORS =    ('satchmo_store.shop.context_processors.settings',
                                  'django.core.context_processors.auth',
                                  'django.contrib.messages.context_processors.messages',
                                  )

INSTALLED_APPS = (
        'django.contrib.sites',
        'django.contrib.messages',
        'satchmo_store.shop',
        'django.contrib.admin',
        'django.contrib.auth',
```

```
            'django.contrib.admindocs',
            'django.contrib.contenttypes',
            'django.contrib.comments',
            'django.contrib.staticfiles',
            'django.contrib.sessions',
            'django.contrib.sitemaps',
            'registration',
            'keyedcache',
            'livesettings',
            'l10n',
            'sorl.thumbnail',
            'satchmo_store.contact',
            'tax',
            'tax.modules.no',
            'tax.modules.area',
            'tax.modules.percent',
            'shipping',
            'product',
            'product.modules.configurable',
            'product.modules.custom',
            'product.modules.downloadable',
            'product.modules.subscription',
            'payment',
            'payment.modules.dummy',
            'payment.modules.giftcertificate',
            'satchmo_utils',
            'app_plugins',
            )

AUTHENTICATION_BACKENDS = (
    'satchmo_store.accounts.email-auth.EmailBackend',
    'django.contrib.auth.backends.ModelBackend'
    )

#### Satchmo unique variables ####
#from django.conf.urls import patterns, include
SATCHMO_SETTINGS = {
                'SHOP_BASE' : '',
                'MULTISHOP' : False,
                'SSL'       : False,
                #'SHOP_URLS' : patterns('satchmo_store.shop.views',)
                }

# Load the local settings
from local_settings import *
```

Note: In order for the admin site to work properly, you must have satchmo_store.shop placed before django.contrib.admin

2. Copy the `local_settings` file to mystore:

```
cp /home/user/src/satchmo-trunk/satchmo/projects/base/local_settings.py \
    /home/user/src/mystore/local_settings.py
```

3. You will need to verify the values assigned to the following items in `local_settings.py`:

```
SITE_NAME
CACHE_BACKEND
```

```
CACHE_TIMEOUT
SITE_DOMAIN
LOGDIR
LOGFILE
```

Note: Satchmo requires that your database be able to support utf-8 characters. This is especially important for MySQL. If you are using MySQL, you may want to use the following statement in your settings file to enforce utf-8 collation:

```
DATABASES['default']['OPTIONS'] = {
    'init_command' : 'SET NAMES "utf8"',
    }
```

Note: If you are using a Windows system, we recommend setting your MEDIA_ROOT using normalize_path as shown below:

```
from satchmo_utils.thumbnail import normalize_path
MEDIA_ROOT = normalize_path(os.path.join(DIRNAME, 'static/'))
```

2.3.2 Configure the rest of the required files

1. Next, you need to configure your `urls.py` file. The most simple `urls.py` file would look like this:

   ```
   from django.conf.urls import *
   from satchmo_store.urls import urlpatterns
   ```

2. If you have additional urls you would like to add to your project, it would look like this:

   ```
   from django.conf.urls import *
   from satchmo_store.urls import urlpatterns

   urlpatterns += patterns('',
       (r'test/', include('simple.localsite.urls'))
   )
   ```

3. Ensure that you have a template directory setup. You only need to place templates in the directory if you are overriding existing templates.

After you have installed everything, you should have a directory structure that looks similar to this:

```
mystore
    |-- __init__.py
    |-- local_settings.py
    |-- manage.py
    |-- satchmo.log
    |-- settings.py
    |-- simple.db
    |-- static
    |   |-- css
    |   |   |-- blackbird.css
    |   |   |-- jquery.autocomplete.css
    |   |   `-- style.css
    |   |-- images
    |   |   |-- blackbird_icons.png
    |   |   |-- blackbird_panel.png
    |   |   |-- productimage-picture-default.jpg
```

```
|   |    |-- productimage-picture-default_jpg_85x85_q85.jpg
|   |    `-- sample-logo.bmp
|   `-- js
|       |-- blackbird.js
|       |-- jquery.ajaxQueue.js
|       |-- jquery.autocomplete.js
|       |-- jquery.bgiframe.js
|       |-- jquery.cookie.js
|       |-- jquery.form.js
|       |-- jquery.js
|       |-- satchmo_checkout.js
|       |-- satchmo_pay_ship.js
|       |-- satchmo_product.js
|       `-- satchmo_store.js
|-- templates
`-- urls.py
```

2.3.3 Test and Install the Data

1. Now, you should be ready to go. In order to test your Satchmo setup, execute the following command (from the mystore directory):

   ```
   python manage.py satchmo_check
   Checking your satchmo configuration.
   Your configuration has no errors.
   ```

2. If any errors are identified, resolve them based on the error description.

3. Sync the new satchmo tables:

   ```
   python manage.py syncdb
   ```

4. Load the country data stored in the l10n application:

   ```
   python manage.py satchmo_load_l10n
   ```

5. (Optional) Load the demo store data:

   ```
   python manage.py satchmo_load_store
   ```

6. (Optional) Load the US tax table:

   ```
   python manage.py satchmo_load_us_tax
   ```

2.4 Basic Configuration

2.4.1 Settings configured in Python files

If you followed the installation steps, you should have a basic store to start using. There are a number of places you might want to configure.

1. In the `settings.py` (or `local_settings.py`) file, there are a number of general Django settings. However, there are a few that are specific to Satchmo. These default Satchmo settings can be overridden by adding them to a `SATCHMO_SETTINGS` dictionary like this:

```
SATCHMO_SETTINGS = {
    'SHOP_BASE': '/shop',
    'MULTISHOP': False,
    'PRODUCT_SLUG': 'items',
    'SSL':   True,
}
```

Satchmo recognises the following keys in `SATCHMO_SETTINGS`:

'SHOP_BASE'

> **default** '/shop'

> Used as the prefix for your store. Don't append a trailing slash ('/') - Satchmo does this for you. In the default setting, your store is located at *www.yourname.com/shop/*. If you would like to change this setting, this is the place to do it. If you would like your store to be at the root of the url, set this to `""`.

'SHOP_URLS'

> **default** []

'MULTISHOP'

> **default** False

> A boolean used to enable or disable an optimization for running multiple sites with independent carts from the same Satchmo instance. If enabled, Satchmo will patch `Site.objects.get_current()` to dynamically determine the current site by looking up the host in the request headers. Normally, you would have to create a separate settings file for each site, which in turn would require loading multiple copies of Satchmo into memory. This optimization can reduce the memory required to serve multiple sites in a memory constrained environment.

'CUSTOM_NEWSLETTER_MODULES'

> **default** []

> A list of custom newsletters.

'CUSTOM_SHIPPING_MODULES'

> **default** []

> A list of custom shipping modules outside of the standard Satchmo distribution.

'CUSTOM_TAX_MODULES'

> **default** []

> A list of custom tax modules outside of the standard Satchmo distribution.

'COOKIE_MAX_SECONDS'

> **default** 60*60*24*30

> Cookie expiration time.

'CATEGORY_SLUG'

> **default** 'category'

> The prefix used for category urls; see satchmo_category and satchmo_category_index.

'PRODUCT_SLUG'

> > > **default** `'product'`

> > The prefix used for product urls; see `satchmo_product`.

> > `'SSL'`

> > > **default** `'False'`

> > Whether or not SSL should be enabled for the checkout modules.

> > `'DOCUMENT_CONVERTER'`

> > > **default** `'shipping.views.TRMLDocument'`

> > Which document generation system to use. See *Setting up document generation*.

2. In addition to the Satchmo specific settings, there are some Django settings you will want to make sure are properly set:

   - Make sure that your `DATABASES['default']['ENGINE']` variable is also set correctly.

   - You should ensure that all of your paths are setup correctly. Key ones to look at are:

     - `MEDIA_ROOT` this is where product images and other uploadable will be stored

     - `MEDIA_URL` '/media/'

     - `STATIC_ROOT` directory for css, js and template images, where they are collected

     - `STATIC_URL` '/static/'

     - `STATICFILES_DIRS` additional directories where search the source static files

     - `ADMIN_MEDIA_PREFIX` (obsoleted with Django 1.4)

     - `TEMPLATE_DIRS`

   Satchmo uses for media files and destination static files different directories, as it is required by Django, but the source directories are for backward compatibility the same: project...directory/static/ However directories can be easy set different now and it is better for the future.

Changing the L10N Settings

Satchmo supports a setting `L10N_SETTINGS` that can be defined in your store's `settings.py` file. To configure the currency format and other internationalization options. The example below would configure the Euro:

```
L10N_SETTINGS = {
  'currency_formats' : {
     'EURO' : {'symbol': u'€', 'positive' : u"€%(val)0.2f", 'negative': u"€(%(val)0.2f)",
              'decimal' : ','},
  },
  'default_currency' : 'EURO',
  'show_admin_translations': False,
  'allow_translation_choice': False,
}
```

Satchmo recognises the following keys in `L10N_SETTINGS`:

> `'default_currency'`

> > The default currency type to display.

> `'show_admin_translations'`

> > **default** `True`
> >
> > Enable or disable the use of the translation options in the admin.
>
> `'allow_translation_choice'`
>
> > **default** `True`
> >
> > Enable or disable the translation section for the store user.

The `L10N_SETTINGS` variable also allows you to control whether or not translation fields should be displayed in the admin. In the example above, they will be disabled. The default is `True`

Setting up document generation

Satchmo is able to generate a various range of *documents* that generally need to be printed on plain old paper or on an equivalent electronic format (such as PDF (Portable Document Format)).

Examples of these documents are invoices, packing slips and shipping labels (although the system can be adapted to much more than this).

For new projects, Satchmo renders these documents as normal HTML pages, which have been slightly optimized for direct printing (i.e. from the browser).

Should this not be enough for you, Satchmo also includes two (optional) ways to export said documents in PDF: one based on trml2pdf and the other based on _wkhtmltopdf.

Which system to use is controlled by the `'DOCUMENT_CONVERTER'` key within the `SATCHMO_SETTINGS` dictionary, which should contain the full dotted path of the *document converter* to use.

wkhtmltopdf

wkhtmltopdf is a program that contains a statically compiled, headless WebKit rendering engine (which is the basis of succesful browsers such as Safari, Google Chrome, and the default browsers on the two most popular mobile platforms).

Installing it is as simple as downloading and unzipping the release for the platform you are using. After doing that, keep note of the directory where you have unzipped the release, and then enable it in your settings file.

To do the latter, set the `'DOCUMENT_CONVERTER'` key within `SATCHMO_SETTINGS` dictionary to `'shipping.views.WKHTMLDocument'`. Then you must tell Satchmo where the wkhtmltopdf binary can be found, by adding a `'WKHTML2PDF_BINARIES'` key in the dictionary, whose value is another dictionary where the keys are the operating system categories as reported by `sys.platform` and the values are the absolute paths to the corresponding wkhtmltopdf binary.

Note: You are **not required** to fill in a key for each platform: the key of the platform you are on is sufficient.

The values for the keys can be obtained by looking at the documentation for the core module `sys`, but here is a quick, non authoritative and incomplete list:

- `linux2` for Linux (`linux` on Python 3.3 and above)
- `darwin` for Mac OS X
- `win32` for Windows

Therefore, if you are on a Linux machine you will do:

```
SATCHMO_SETTINGS = {
    ...
    'DOCUMENT_CONVERTER': 'shipping.views.WKHTMLDocument',
    'WKHTML2PDF_BINARIES': {
        'linux2': '/path/to/wkhtmltopdf-i386'
    },
    ...
}
```

While on Mac OS X you will do:

```
SATCHMO_SETTINGS = {
    ...
    'DOCUMENT_CONVERTER': 'shipping.views.WKHTMLDocument',
    'WKHTML2PDF_BINARIES': {
        'darwin': '/path/to/wkhtmltopdf'
    },
    ...
}
```

trml2pdf

If you wish to use trml2pdf, you must first install Reportlab (based on the description for your OS here) and then trml2pdf itself:

```
pip install http://www.satchmoproject.com/snapshots/trml2pdf-1.2.tar.gz
```

Then you must set the `'DOCUMENT_CONVERTER'` key within `SATCHMO_SETTINGS` dictionary to `'shipping.views.TRMLDocument'`.

Customizing templates

If you wish to customize templates, keep in mind that they are located in `shop/docs/html` within the `shop` application for HTML templates (used by the default HTML generator, and by the wkhtmltopdf one) and within `shop/docs/rml` for the trml2pdf ones.

Developing your own

If you wish to develop your own *document converter*, take a look at `shipping.views` and implement a class with the same methods as `HTMLDocument`.

You will then be able to activate it by setting `'DOCUMENT_CONVERTER'` to your class *full dotted name* (e.g. `'my.module.converter.MyDocument)`

Note: If you use a unicode character, you'll need to have an encoding at the top of your settings.py file:

```
# -*- coding: UTF-8 -*-
```

2.4.2 Settings configured via Django's admin interface

The majority of the store configuration is done through the admin interface. This can be accessed from the main admin page (usually at */admin/*) via the *Admin → Edit Site Settings* link. It is also usually available at the URL */settings/*.

All of the configuration settings have detailed help notes. They also default to sensible configurations so your initial store should work fine without changing any values.

Base Settings

These items are used for general store configuration and include:

- Account verification options
- Default currency symbol
- Enable/disable product ratings
- Controlling display of featured products
- Controlling quality of thumbnail creating
- Enabling sending of html formatted emails

Google Settings

This section allows you to enable or disable google analytics and conversion tracking for adwords.

Payment Settings

Satchmo can handle multiple ways of accepting payment. By default, you have a dummy processor that does nothing but accept payments. Obviously, you'll want to enable one of the other modules before going live.

Each payment module will have it's own configuration items. These items apply universally to all payment modules.

- Accept real payments
- Allow URL access for cron rebilling of subscriptions
- Force ship to and bill to countries to match during checkout
- Cron passkey to allow subscription rebilling

> **Warning:** After saving changes to your payment processor, you will need to restart your server for the changes to take effect.

Product Settings

Before you use any of the products, you need to make sure the appropriate products are added to your `INSTALLED_APPS`.

In this section you can also configure:

- Allowing checkout with 0 inventory
- Using Akismet for comment spam prevention
- Number of recent items displayed
- Measurement system
- Number of featured items
- Random display of featured products

- Protected directory to be used for downloadable products
- Specific directory where images should be uploaded

Shipping Settings

This section allows you to choose which shipping modules you want to make available to users when they check out.

Once you select the modules you would like to use, you will be given an option to enter any additional information required for that module.

Tax Settings

Satchmo allows different tax configurations. This section allows you to choose the active tax module and configure it for your store.

Newsletters

Satchmo has two methods for handling newsletter subscriptions. By default, you have an "ignore it" processor enabled. To enable handling, first add `"satchmo.newsletter"` to your list of installed modules in your settings file.

Next, choose the way you want to handle the subscriptions. Currently we have two working newsletter plugins:

- `satchmo.newsletter.simple` - This just tracks subscriptions in a database table for your querying pleasure. You can then export that list to whatever mailing manager you want to use.
- `satchmo.newsletter.mailman` - This is an integration module which works with Gnu Mailman (http://www.gnu.org/software/mailman/). This is particularly convenient if you have a Cpanel VPS system, since Mailman is installed by default on most such systems. To use this, you need to make sure Mailman is on your PYTHONPATH and you should have already set up a mailing list as an announce-only list (http://www.modwest.com/help/kb13-195.html). You'll need to enter the name of the list in your local settings file.

SSL

SSL Security can be set on any url in your store. In order for SSL to work, make sure that it is enabled in the middleware section of your settings.py:

```
MIDDLEWARE_CLASSES = (
    "django.middleware.common.CommonMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.locale.LocaleMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.middleware.doc.XViewMiddleware",
    "satchmo.shop.SSLMiddleware.SSLRedirect"
)
```

In order to support a fully encrypted page, you also need to make sure you provide a secure url for the media. This url will automatically be used in pages served by SSL, but only if you specify it in your settings.py:

```
MEDIA_SECURE_URL = 'https://secure.example.com/static/'
```

Then, enable it for the specific urls you would like to be protected by adding `{'SSL': True}` to each url. Here's an example which would enable SSL for login:

```
(r'^accounts/login/$', 'login', {'SSL': True, 'template_name': 'login.html'}, 'satchmo_login'),
```

SSL for Payments works slightly differently. The are controlled by the Satchmo setting described above. To have all checkout pages enabled for SSL, just set `SSL:True` in your Satchmo settings.

Disabling the Live Settings System

Once your store is live, you may want to disable the admin ability to edit the site configuration. To do this, edit your settings.py file and add a new entry `LIVESETTINGS_OPTIONS` with the settings you want to lock into place. The easiest way to get it is by the URL http://your.site/settings/export/.

The `LIVESETTINGS_OPTIONS` must be formatted as follows:

```
LIVESETTINGS_OPTIONS = {
    1 : {
        'DB' : False,    # or True for use db and ignore these settings
        'SETTINGS' : {
            'GROUPKEY' : {
                'KEY' : val,
                'KEY2' : val,
                # ...
            },

            'GROUPKEY2' : {
                'KEY' : val,
                'KEY2' : val,
                # ...
            },

            # ...
        }
    },

    # ...
}
```

In the settings dict above, the `1` is a site index, allowing you to have different settings for different sites. The `val` entries must exactly match the format stored in the database for a setting. For example, do not use a literal `True` or an integer, it needs to be the string representation of them.

If `DB` is `False`, then editting the settings via the admin will be disabled. All configuration must then be done through the settings file.

The easiest way to do this is to query the database for livesettings_setting and livesettings_longsetting, and convert to a Python dictionary manually.

Store Configuration

The final configuration option that is available is configuring which Countries you would like to ship to. This option is available through the Admin interface through *Admin → Shop → Store Configuration*. It is typically accessed through the URL */admin/shop/config*

This section allows you to fill in store address and basic demographic information that is used throughout Satchmo. The *Shipping Countries* section will allow you to configure:

  • Whether or not to ship only within 1 country

  • The Default Country to ship to

- All Countries which may be chosen during the checkout process

If you choose to allow shipping to multiple countries, the checkout process will automatically populate valid states based on the selected country.

Store Usage

3.1 Products

The core of any online store is the product you are selling. Some people may have very simple needs for categorizing and describing their products, while others may have much more complex needs. Satchmo's product framework tries to strike a balance, enabling easy product configuration but supporting complex products if your needs demand.

3.1.1 Categories

Out of the box, Satchmo supports the traditional hierarchical method for categorizing products. In other words, you can create category trees like this:

- Movies
  - Action
  - Western
  - Comedy
  - Black and White
    * Foreign
    * English
    * Silent
- Books
  - Fiction
  - Non-fiction

Some things to keep in mind with Satchmo's implementation:

- You are not limited to the number of categories or the depth of categories.
- You are allowed to add products to 1 or more product categories

When entering a category in the admin interface, you will have several fields to fill in. Below is a description of each field and how it is used:

> **Name** The name of this specific category. It is the text seen by the user to describe this category. In the example above, Movies, Action, etc. are all names.

Slug This is a prepopulated field that is used in the url to identify the category. One aspect that Django encourages is well designed urls. The slug field is commonly used so that a url "makes sense" when you look at it. The nice side effect of this is that in some cases, search engines will give higher ranking for sites with cleaner urls.

Parent If a category is at the top of the tree, then there is no parent. In our example above, Movies and Books have no parents. While Action, Comedy, Black and White would all have Movies as a parent.

Meta When an HTML page is constructed, certain meta information can be added to the page which makes it easier for search engines to classify your pages and your products. This field is used to enter information about your category that may not be readily seen by just looking at a description.

Description This is a free form field that describes this category. You can use this to help guide users in your site. It's also helpful for search engines.

Ordering This field can be used to specify which order the categories should be displayed.

Active This allows a category to be populated with information but to be set inactive.

3.1.2 Product

A Product is the main focus of your store. It is the "thing" that a person going to your store would see.

To activate the various product modules, they need to be enabled in your settings.py file. For example:

```
INSTALLED_APPS = (
        ...
        'product',
        'product.modules.configurable',
        'product.modules.custom',
        'product.modules.downloadable',
        'product.modules.subscription',
        ...
```

Once the products have been added, there are many fields you can use to configure your Items.

Category See the description above. This is where you choose one or more categories where a person could find this product.

Name This is the full name of an product. Think of it as the title of the product.

Slug This is similar to the slug field in category. It is a brief description of the item that will show up in the url and uniquely identify this product.

Description Free form text fields that tell a potential customer about this item and why they may want to purchase it.

Short_description This is meant for a brief 1 or 2 line description that can be shown by your templates on the product category pages and similar places.

Date_added The date you added this product to your store. It could be useful for showing a list of new products.

Active A flag to identify which items are still valid in the store. If you decide you don't want to show an item for purchase, we recommend making it Active=False instead of trying to delete it.

Featured Another flag which can be used to identify special items that you may want to highlight in your site. In the default templates, featured items show-up on the front page of the store.

Items in stock The inventory of this specific item.

Meta This meta field is similar to the one described above in the category. It's basically some additional descriptive language you might want to include in your HTML that does not already show up in the description.

Weight, Length and Height These dimensions are for your use in displaying the item's info. It can also be used for shipping calculations.

Taxable A flag to tell whether or not tax should be calculated for this product.

Tax_class Allows you to set the rate at which tax is calculated.

Related_items Certain products may naturally be grouped together. If you were to purchase a movie, you might want to include links to the book based on the movie or maybe a movie poster. This would be the place to record those linkages. By default, this does not show up in the store. You will need to modify your templates to display it.

Also_purchased Another field to show relationships between products. This can be useful if you want to try to increase sales by showing what others have purchased when they bought this product.

Price An item has multiple ways to determine a price. For a simple single-price all the time item, just enter a price with a "Discount Quantity" of 1, and a blank "Expires" date. More detailed options are spelled out in the Price and ConfigurableProduct sections below.

Images Add an image to your product, see Images section below.

Product Subtypes There's a variety of ways that the behavior of a product can be expanded. These additional, optional, features are shown as product SubTypes. The most commonly used of these will be ConfigurableProduct which allows you to have options on your product such as Sizes: Small/Medium/Large, Colors, etc. For more details on the Product SubTypes look in the applicable sections below.

Product Attributes

In some cases, you may want to have certain product information stored in the database. For instance, you might want to have "number of pages" or "ISBN" fields for all of the books in your store. Attributes allow you to associate arbitrary data with your products via name, value pairs.

In the latest version of Satchmo, these are more powerful and useful because you can define specific attributes to capture as well as validation to perform on the data.

Refer to *Custom Product Attributes* for more details on using this feature to customize the information you store and present for your products.

Price

The final choice you have with your products is if there is additional pricing you would like to apply for bulk or special discounts. For instance, you may want to charge someone $10 for 1-5 shirts but only $7 if they order 6 or more. This model allows you to do such configuration.

Price Override the price for this sub item if it meets one of the other criteria identified below.

Quantity The number of items needed to qualify for the price change.

Expires Date when this offer is no longer valid. This can be used for various promotions.

Images

In addition to the information discussed above, you can add as many images as you would like to your item. One of the nice things about the way Satchmo handles these images is that they are automatically converted to thumbnails and are cached so that the conversion process does not slow down your web site.

Images only have a couple of fields:

Picture This is the image of your item that you upload to the web server. The actual thumbnail size is set in your templates.

Caption A description of this image. This field may be used to describe the different angles that images show.

Sort If you would like control of the order in which the images are displayed, this field can be used.

3.1.3 Product Options

As discussed above, a product is the central focus of your store. For many cases, the products in your store may have several options. For instance, if you sell t-shirts, a visitor to your store should be able to choose the size of the shirt while looking at the main item. You may also want them to choose a color, style and/or a bunch of other different options. The challenge with this much flexibility is building something that is easy for the user to understand and easy for the store administrator to maintain. Satchmo uses option groups and options to create configurable products and product variants.

Option Groups

Continuing on with the shirt example, you will probably end up with some similar options that you want to apply to a large number of products. For instance, all of your shirts are going to have sizes of Small, Medium and Large. Instead of adding all three of these to each item, you can create an option group and add that grouping to the item. Maintenance is much easier this way.

Here are the fields in the option groups:

Name A description of the group. This is displayed in the product page to describe the choices available to the user. In our shirt instance, "size" would be appropriate.

Description This field is used for administrative purposes. Size may make sense when looking at a product but what if you have shirts, shoes and hats? How do you know which group these sizes apply to? Use this field to write details about the size. For instance, you could have an option group with the description "Shirt Sizes", "Hat Sizes" and "Shoe Sizes."

Sort Order This helps you order the options in the way you would like them displayed. For instance, you may want colors to show before sizes.

Options

In our example, if an option group is size, then the options would be "small", "medium" or "large." The nice thing about these options is that you can configure these to change the price of your product. If a large shirt costs more than a small shirt, then you can assign the incremental cost to the "large" option item and the correct price will be shown in the users cart and order.

Name This is the displayed value of the option. You may want to show people "Small" but only store "S" in the database.

Value Corresponds with the Name above. In this example, the value would be "S."

Price Change If you would like this option to modify the Item's price, enter a positive or negative number here.

Configurable Products

ConfigurableProduct is a modifier for a product that allows you to associate an OptionGroup with a particular Product.

Option group This is a collection of descriptions that identify this item. See more details below.

Create_Variations This is a special option that allows you to create ProductVariations.

Variations Variations are all Products that are never displayed directly to a customer, but represent the actual product as you would have it on the shelf. For example, if your ConfigurableProduct is a t-shirt, then you will probably have ProductVariations for "Large Green t-shirt", "Small Blue t-shirt", etc. It's easiest to have these generated for you using the create_variations option above, but once they are created, you can edit these individually to do things such as set a different price for the XL shirt.

Product Variations

As described above a ProductVariation can be thought of as the actual product you maintain in inventory and sell to a customer. It is an item with all of the options applied. In our example case, it might be a Large, White Shirt. The ProductVariation object itself only has links between a ConfigurableProduct, and a Product.

Downloadable Product

A downloadable product is a virtual product that will be electronically delivered to a customer after the purchase is completed. In Satchmo, this is accomplished by emailing a unique url to a customer after the purchase. This url can be restricted so that the possibility of multiple downloads is minimized. Depending on your product you may want to add additional security controls but that is outside the scope of Satchmo.

In order for the downloadable product to be secured, you will need to ensure that the protected directory can not be browsed to directly; you should also ensure that the directory where the file is stored on the server is protected from other users.

Adding a Downloadable Product

1. Add the dotted name `'product.modules.downloadable'` to `INSTALLED_APPS` in your settings.py.

2. Create the tables by with `python manage.py syncdb`.

3. In the "Downloadable" section of the admin site, add a new instance of a Downloadable Product as you would usually do.

The last screen will allow you to:

- upload your file;

- specify the number of allowed downloads;

- specify the number of minutes it will remain active;

> - a flag for whether or not the file is still active.

After saving the changes, the product will be enabled for downloading.

Storage and Serving Location of Downloadable Products

As Satchmo relies on the default storage class, the files will be stored in `PRODUCT.PROTECTED_DIR` (by default, `"protected"`), a subdirectory of `django.core.files.storage.default_storage.location` (usually, `settings.MEDIA_ROOT`).

The Downloadable Products will be served from `/<PRODUCT.PROTECTED_DIR>/`. To modify this location, please see `satchmo_store.shop.signals.sendfile_url_for_file()`.

Configuration

In the below configuration examples, the following assumptions are made:

- `"/usr/local/www/website/static"` to be `default_storage.location`;
- `PRODUCT.PROTECTED_DIR` to be the default `"protected"`;
- the URL `"/protected/"` to be location from which the products will be served.

Apache   Install mod_xsendfile, and add this to your configuration:

```
<Location /protected/>
    XSendFile on
</Location>
```

Lighttpd   Set `"allow-x-send-file"` to `"enable"` in your FastCGI/SCGI config, and add also the below to your configuration file:

```
$HTTP["url"] =~ "^/protected/" {
    url.access-deny = ("")
    server.document-root = "/usr/local/www/website/static"
}
```

Nginx   Add this to your configuration file:

```
location /protected/ {
    internal;
    root /usr/local/www/website/static;
}
```

Custom Product

A custom product is one that is typically assembled or made to order based on the customer's needs. A common situation is a computer configurator that allows a customer to tailor the computer to their needs. Once the order is submitted the store owner will then build or configure that product.

Once you have created a product as described above, you may wish to use it as a basis for a custom product. The most important additional information you need to associate with your product is the option groups or custom text fields the customer may use to create their product. The option groups are no different from the ones described above. This

gives you tremendous flexibility to vary the price of the product based on the chosen options. The custom text fields are free form text fields meant to capture special notes or instructions related to the order.

One unique aspect of the custom product is that you can elect to charge only a certain percentage at the time the order is placed. If you choose a percent downpayment then only that amount will be charged when the order is completed. If you choose 100%, then the entire price will be charged at checkout.

The final unique aspect is that you can elect to defer shipping charges on the order. If you are creating a very unique product, you may wish to have that discussion with the customer when you are collecting the full payment.

The custom product does not create product variations. The downside of this is that you must manage your inventory yourself but the upside is that you don't have all of those different configurations in your store. Feel free to play around with both products and see which one meets your needs the best.

Subscription Product

A subscription product is a product type that can be used to manage recurring billing memberships or to add payment terms to a non-membership product.

In order to use this product,make sure you have the Subscription Product type enabled in the configuration settings. Additionally, if you would like to use a url to activate the rebilling, you will need to make sure the the setting ALLOW_URL_REBILL is set to True. If you do set it to True, make sure that you add a new unique key in the CRON_KEY setting.

To use a subscription product, you need to setup a base Product as described in the sections above. The price you set for the base product will be the amount that is charged to your customer periodically based on your subscription schedule

In order to use this product type, from the Product detail page, choose Add Subscription Product.

On this screen, you will enter your subscription payment terms as described below.

> **Recurring Billing** Select this if you want your customer to be charged the regular product price on a repeating schedule
>
> **Recurring Times** Enter an integer here to limit the number of times your customer will be charged. (ie only 10 easy payment of $9.95) If this is a perpetual subscription, enter 99999.
>
> **Duration** Enter the number of units between between each billing cycle.
>
> **Expire Unit** This is used in combination with the duration to determine whether the timing should be in months or days. For example, if you want to bill monthly for a year, you could set it to a duration of 12 and Expire Unit of monthly.
>
> **Shippable** The shipping charges, if any, that will be charged to your customer.
>
> **Trial Terms** Use this section to add trial pricing to your product (ie only $4.95 for the first 7 days, then $29.95 a month thereafter).
>
> > **Price** The price of the trial period. Enter 0 for a free trial, or a decimal number for a non-free trial (ie $9.94)
> >
> > **Trial Duration** The number of days the trial will last. Leave both price and trial duration blank if you do not want to add a trial.

Note: You can add as many trial periods as you wish, unless you are using paypal as a payment option. PayPal can only accept 2 trial periods. If you are using PayPal, do not add more than 2 trial periods!

Cronjob:

In order for recurring billing to work, you need to setup a cronjob on your server to run once a day (preferably in the middle of the night). There are a couplf of options to do this.

Using satchmo's custom management command:

```
0 23 * * * python manage.py satchmo_bill_recurring
```

Using lynx:

```
0 23 * * * /usr/bin/lynx -source http://yourdomain.com/shop/checkout/cron/?key=YOURPASSKEY
You will need to set the CRON_KEY variable in your settings file to your desired passphrase.
```

Using a shell script:

```
0 23 * * * sh /path/to/satchmo/rebilling_cron.sh
```

If you use this method, make sure to edit the path on the first line of rebilling_cron.sh to point to the directory containing the script.

Gift Certificate Product

A gift certificate is a special product. In order to use it, make sure it is an enabled product type and that it is a selected payment method.

Once you have enabled the gift certificate product type and payment methods, you need to create a product as described above. Then, in the product subtypes section, select "Enable GiftCertificateProduct" to turn the product into a gift certificate. Now, a user has the option of purchasing a gift certificate and receiving a unique code that can be applied towards the purchase of a product at your store.

3.1.4 A short tutorial

For a more detailed tutorial that covers how to add a product, please review this *tutorial*.

The product variation manager contains a set of helper functions for managing product variations. It is accessed at /product/admin/variations/ and provides a streamlined process for managing all of your configurable products.

3.2 Discounts

Satchmo allows you to setup multiple types of discount codes:

- Straight dollar amounts
- Percentage off
- Free shipping

Each of these types can be qualified or limited by:

- Number of uses
- Minimum order
- Specific dates
- Specific products

3.2.1 Creating Discounts

A discount code is easy to setup in Satchmo using the following fields:

Description A simple characterization of the discount. For example, "New User Discount" or "Repeat Customer Promotion."

Code The unique identifier a user would enter to activate this discount.

Amount A straight dollar discount off a product. You can only have a dollar OR percentage.

Percentage A percent discount off the price.

Auto Discounts Use this field to advertise the discount on all products to which it applies. Generally this is used for site-wide sales. See *Site-wide Sales via Auto Discounts* for details.

Allowed Uses The number of times this discount can be applied.

Num Uses The number of times this discount has been used.

Minimum Order The minimum amount that must be purchased before this can apply.

Start Date The date when this discount becomes effective.

End Date The date when this discount is no longer valid.

Active A flag you can set to make it active. If you want to create a discount in advance but hold off on making it available, you can use this flag.

Free Shipping Whether or not this discount allows the user to not have to pay shipping charges.

Include Shipping A flag to determine if shipping costs should be affected by the discount.

Valid Products A list of all the products this discount is valid for.

Valid Categories New in version 0.9.1.

A list of categories this discount is valid for; the discount will be valid for products in these categories, as well as in their child categories.

3.2.2 Site-wide Sales via Auto Discounts

Automatic discounts are how you do side-wide sales.

> **Warning:** Automatic discounts must always be percentage sales, and you can only have one running at a time. You should also ensure that you do not apply auto discounts to gift certificates.

1. In your site's admin interface, create a new discount.

2. Make an "automatic" discount, so that it is available to the automatic discount template tags. Do not enter a new price in the admin for the products.

3. In the template where you want to show the auto discount, load the `{% satchmo_discounts %}` (Visit *Satchmo Templates* documentation for details on Satchmo filters) filter library.

4. You can now add some markup to your template, similar to below:

```
{% if sale %}
<div>
  <div class="saleprices">
    <h3> Regular price: <span class="saleprice" id="price">{{
```

```
                          product.unit_price|currency}}</span></h3>
      <h3> {{ sale.percentage_text }} off: <span class="saleprice" id="sale_saved">{{
                          product|discount_saved:"sale"|currency }}</span></h3>
      <h3> Your price: <span class="saleprice" id="sale_price">{{
                          product|discount_price:"sale"|currency }}</span></h3>
    {% else %}
      <h3 id="price">{{product.unit_price|currency}}</h3>
    </div>
    {% endif %}
```

The key parts of the above markup are the `{{ product|discount_saved:"sale"|currency }}` and the `{{ product|discount_price:"sale"|currency }}` filtered variables. The former returns the amount saved by the discount while the latter returns the discounted price, all in the correct locale currency.

3.3 Pricing

Satchmo has flexible pricing that allows you to price at multiple levels.

3.3.1 Pricing at the Item Level

You can create top-level Products, which are the things your customers see on the website. (Ex: Python Rocks shirt in demo store) Additionally, these products can have options (Ex: sizes, colors, etc). The product has a variation for each combination of options. This can be used to track inventory of particular product variations. (Ex: The Large, Blue, Python Rocks Shirt)

Effectively it works like: Product + Option = Variation

Pricing fits into the picture this way:

   • You can set a default price on your Product.

   • You can set a price adjustment for an option.

   • You can set a price override for a variation.

The price code looks through these backwards. If the variation has a price then that is used Otherwise the Product price is used +/- the adjustment for the chosen options (IE: XL shirts are +$1.00)

With this capability, you can quickly price a simple store by just using the product prices, and possibly adding price adjustments for specific options, but you have the option of very fine-grained price control by setting prices for each variation.

3.3.2 Product Variation Prices

On the product variations, the price can have an expiration date and/or a quantity. If either of these fields are set then that price is only effective for orders placed before that date, or >= that quantity. The most specific of these gets highest priority (IE: the soonest expiration date, and the highest applicable quantity). This is useful for running temporary promotions that automatically expire.

3.3.3 Expiring Pricing

If you are using expiring dates for prices, please note that you must run a daily update of the pricing lookup table. To do this, you need to run **./manage.py satchmo_rebuild_pricing**

This command also helps every time when this error occurs: "Sorry, we don't have any of that combination available."

The easiest way to run this automatically is to use the excellent "Django_Extensions" app. We've already added a daily job to rebuild prices, so all you need to do is to ensure that the daily job is getting run via crontab:

```
# run every morning at 2:22 am
22 2 * * * /usr/bin/python /path/to/site/manage.py runjobs daily >/dev/null 2>&1
```

Alternatively you can directly execute the command via crontab:

```
# run every morning at 2:22 am
22 2 * * * /usr/bin/python /path/to/site/manage.py satchmo_rebuild_pricing >/dev/null 2>&1
```

3.3.4 Pricing Tiers

Satchmo supports setting different price and discount tiers based on user groups. The most common reason a store owner may need this is if they want to offer different discounts or prices for a class of user. For instance, a user with a "Gold" membership may automatically get a percentage discount. Another common usage is for wholesale versus retail pricing.

This feature is optional and can be enabled by following these steps: Using it is quite simple:

1. Add 'satchmo_ext.tieredpricing' to your INSTALLED_APPS.

2. Run 'manage.py syncdb'.

3. In the admin site under 'Auth > Groups', create a user group for your desired tier.

4. Add users to that group via 'Auth > Users' in the admin.

5. Make a PricingTier, associating the group to the tier, and setting any default percentage discount.

6. If you want specific tiered pricing for a product, then edit the product. You'll see a new section, "Tiered Prices", where you can set prices by tier.

The price resolution process is straightforward.

1. If the user is anonymous or has no groups, use the non-tiered price.

2. If the user has a group and that group has a tier:

   2a. Look for explicit prices (see #6 above) for that product. If found, return it. 2b. Else return the non-tiered price reduced by the default tier discount percent.

3. If the user has multiple tiers, return the lowest amount found in 2a & 2b.

Pricing tiers should not be applied to superuser or staff members and they are ignored for them. It is easy to create other unprivileged user account for testing or shopping.

3.4 Tax

3.4.1 Tax Module Activation

Satchmo provides you the flexibility to enable several different types of tax modules.

To activate the various tax modules, they need to be enabled in your settings.py file. For example:

```
INSTALLED_APPS = (
        ...
        'tax',
        'tax.modules.no',
        'tax.modules.area',
        'tax.modules.percent',
        'tax.modules.us_sst',
        ...
```

Note: You must include tax as well as the specific module under tax that you would like to enable.

Activate and configure your tax modules from your *Site Settings page*.

Note: If you have custom tax module you would like to use, add the module's dotted name to *CUSTOM_TAX_MODULES*.

3.4.2 Available Modules

Satchmo includes several tax modules in the default distribution. In general, you will want to enable only one of these. However, based on your products and your geography you will need to figure out what makes sense.

No

This is a very simple module which does not tax any product. It is meant as an example and will probably not be useful in a production store.

Area

This tax module allows you to set taxes based on the geographic area a shipment is being sent to. For example, in the US, this would be used to set varying sales tax rates based on the State.

The satchmo admin command satchmo_load_us_tax will load sales tax rates for each state. You should verify these and make sure they are correct.

The admin also includes the option to calculate the tax based on the shipping or billing address.

Percent

This simple module will charge a flat rate percentage (configured in the admin) to every purchase.

US Streamlined Sales Tax (SST) Processing

The SST module is a complex application that uses the SST service to try to streamline sales tax processing in the US.

> **Warning:** After changing tax modules, you must restart django to enable the new module.

3.5 Shipping

3.5.1 Configuration in Site Settings

The shipping settings are found on the *Site Settings page* under *Shipping Settings*. The settings that apply across all modules are listed below.

Note: Settings specific to the *Flat* and *Per* modules are also displayed in this section.

SHIPPING.**HIDING**

> **Description** Hide shipping form fields if there is only one module available
>
> **Choices**
>
> > - Yes
> >
> > - No
> >
> > - Show description only
>
> **Default** No

If your store only has one shipping option, then you may prefer not to show any ability to select shipping options.

SHIPPING.**MODULES**

> **Description** Active shipping modules
>
> **Choices**
>
> > - *Dummy*;
> >
> > - *Flat*;
> >
> > - *Per*;
> >
> > - *Canada Post*;
> >
> > - *Fedex*;
> >
> > - *UPS*;
> >
> > - *USPS*;
> >
> > - and the modules listed in *CUSTOM_SHIPPING_MODULES*.
>
> **Default** *Per*

Satchmo provides you the flexibility to enable as many shipping modules as you would like. If you have custom modules you would like to use, see Custom Modules below.

SHIPPING.**SELECT_CHEAPEST**

> **Description** Select least expensive by default
>
> **Choices**
>
> > - *True*
> >
> > - *False*
>
> **Default** *True*

A Boolean value.

3.5.2 Enabling Modules

1. Under the *Shipping Settings section* on the *Site Settings page* , select the module you want in `SHIPPING.MODULES`.

2. Save.

3. You should see a new configuration section for the shipping module you selected earlier; open up the section and configure the module accordingly.

Note: Unlike most modules, settings for the default Satchmo modules *Flat* and *Per* modules are not displayed in a separate section; they are displayed alongside settings that apply across all modules in the *Shipping Settings section*.

Custom Modules

Note: You need to go through these steps before going through the instructions in Enabling Modules.

1. Add the module's dotted name to *CUSTOM_SHIPPING_MODULES*.

2. If the module has Django-style models, proceed to *Modules with Django-style Models*; otherwise, proceed with the instructions in Enabling Modules.

Modules with Django-style Models

Note: You need to through these steps before going through the instructions in Enabling Modules.

In addition to the instructions above, you'll have to create the database tables for these modules:

1. Add the module's dotted name to *INSTALLED_APPS* in your settings.py.

2. Run *python manage.py syncdb*.

3. Proceed with the instructions in Enabling Modules.

3.5.3 Generic Modules

Dummy

> **dotted name** `shipping.modules.dummy`

This module is mainly included in order to demonstrate how you can create your own modules. If you have an interest in creating your own, copy this module to a new directory and use the comments to guide you through the process.

Flat

> **dotted name** `shipping.modules.flat`

This is a very simple module that allows you to set a flat rate for all shipments. It's not very sophisticated but it may be useful for simple stores or as a basis for other modules. The values can be configured through your settings page.

To enable, see Enabling Modules.

Note: Unlike most modules, settings for this module are not displayed in a separate section; they are displayed alongside settings that apply across all modules in the *Shipping Settings section*.

No Shipping

Per

> **dotted name** `shipping.modules.per`

This module is a little bit more complex than the flat rate. It allows you to set one rate that is multiplied by the number of items in your order. These values are configured through your settings page.

To enable, see Enabling Modules.

Note: Unlike most modules, settings for this module are not displayed in a separate section; they are displayed alongside settings that apply across all modules in the *Shipping Settings section*.

Shipping By Product

> **dotted name** `shipping.modules.productshipping`

Tiered

> **dotted name** `shipping.modules.tiered`

This one is much more flexible than any of the prior modules, so it requires a little more configuration. It is not enabled by default.

To enable, see *Modules with Django-style Models*.

Why bother?

Well, it allows you to set up multiple carriers, for one thing. For another, you can have different prices based on cart total (total of shippable items, to be specific). For another, it is multilingual from the start, with you being able to specify translations of carrier, method, etc. right in the admin pages. Lastly, you can have shipping specials, expiring on the dates of your choice.

For example, you can make this shipping table:

Price range of cart items	UPS	Fedex
$0-$25	$8.50	$18.00
$25.01-$50	$9.75	$21.00
...	...	...
>$250.00	free	free

Tiered Weight

> **dotted name** `shipping.modules.tieredweight`

This module provides similar features to *Tiered* and requires the same extra steps to install. In addition you can specify a number of zones for each carrier adding a number of countries to each zone. You can then choose a default zone for each carrier. Each zone can have a number of cart weights with corresponding price and handling fee. This means that at checkout the shipping price is based on the shipping country and the total weight of the cart.

NOTE: the weight value of the Weight Tier part of the Zone object is called "Min Weight", but each tier is used for carts that weight at MOST that amount. This means this is the MAXIMUM weight that will be handled by that price.

Tiered Quantity

> **dotted name** `shipping.modules.tieredquantity`

3.5.4 Specialised Modules

UPS

> **dotted name** `shipping.modules.ups`

To enable, see Enabling Modules.

The UPS shipping module provides an interface to the UPS Rating and Service selection interface. This service from UPS allows you to get custom real time shipping quotes based on the sending and receiving addresses as well the items in the shipment.

The Satchmo module uses the XML interface which provides maximum flexibility. The UPS interface is very robust and allows many complex actions. Satchmo is configured to support a basic configuration. If you choose to use this service, you should review the UPS developer documents (available when you sign up) and verify that the shipping configuration currently used by Satchmo makes sense for your needs.

Before you attempt to test the UPS module, you must sign up for a developer account at the UPS web site.

Make sure that your store mailing address is correct in the Store Configuration too.

You must configure the following settings in the Site Settings -> UPS Shipping Settings:

- Type of packaging used by your store (refer to UPS docs for details)
- Your UPS account number
- UPS Pickup option
- Shipping choices you wish to offer to your customers
- UPS user ID
- UPD user password
- UPS XML Access key

Refer to the UPS developer documentation for more definitions on these options.

The UPS api is very powerful and does extensive validation on the data being submitted. Depending on your configuration, a number of possible errors could be returned by the UPS servers. For many of these errors, Satchmo chooses to silently ignore the UPS option that has an issue. The design philosophy is that it is better to not present a price than to present one that may be inaccurate. For this reason, it is strongly encouraged that you have at least one other shipping module enabled so that if there is an error, your customer will still be able to successfully complete the checkout process. To view any errors that may have been generated by the UPS module, please refer to the satchmo log stored in the location specified in your local_settings.py file.

The shipping module does rely on the weight and dimension data you have entered for your products. If you have not entered the weight, then the module will not display any choices.

One important note to remember with the UPS module is that when a user has multiple products in the cart, the UPS request will quote this as if each product is shipped in a separate package. For example, two 3 pound packages will cost more to ship than one 6 pound package. If your shop has a different method of shipping, you will need to override the /apps/shipping/templates/shipping/ups/request.xml file to group products together into one package.

Fedex

> **dotted name** `shipping.modules.fedex`

To enable, see Enabling Modules.

The Fedex module allows Satchmo to calculate shipping costs using the Fedex Web Services interface. The Fedex module has two additional dependencies:

1. python-fedex
2. suds

> **Warning:** The Fedex module will not work correctly without these modules installed on your system.

One important note to consider is that each product MUST have a weight of at least .1 pounds for Fedex to give a valid response. Therefore, there is a setting in the Fedex module to allow you to set a minimum weight that will be used if the individual products do not have weights.

Once you've got the module installed its time to get your account set up with FedEx. (It does not work without the following steps)

1. Go to http://www.fedex.com/us/developer/
2. Log in (you may need to create an account) and go to technical resources. https://www.fedex.com/wpor/web/jsp/drclinks.jsp?links=index.html
3. Click "Get Started with FedEx Web Services Technical Resources now"
4. Click "Move to Production"
5. At the bottom click "Get Production Key"
6. Answer the questions / Fill out the forms. Note: You will need a FedEx account number.
7. Save your authentication key and meter number.
8. Add your meter number to the shop settings after enabling the FedEx shipping module.
9. Check the box to connect to the production server.

USPS

> **dotted name** `shipping.modules.usps`

To enable, see Enabling Modules.

The United States Postal Service module allows you to calculate shipping costs using the USPS web api. Here are a couple of noted regarding this module:

- Specify your username and password in the configuration section for USPS in *Site Settings*.
- The USPS API won't return shipping rates if you are trying to go against the testing server, so a couple of the settings in modules/config.py become useless and confusing
- Make sure your products have weights associated with them. The USPS API won't accept weightless packages and your list of shipping options might not be a list at all!
- The module assumes that the value entered in a product's "weight" attribute is in pounds (because that's what USPS expects)
- Some rates such as parcel post automatically include the "additional postage may apply" values, and according to exchanges with the USPS support crew, we just have to deal with it.

Canada Post

> **dotted name** `shipping.modules.canadapost`

To enable, see Enabling Modules.

Prerequisites

- You must specify dimensions and weight for the products you intend to ship using Canada Post.

- You must have a merchant id assigned by Canada Post to use the module in production environment. Visit http://www.canadapost.ca/business/default-e.asp for more information.

- You must fill out the "Home > Shop > Store Configurations" information and most specifically the postal/zip code.

Notes

- Turn around time is the average time that you will take to process an order before it is ready for Canada Post to pick it up. It may include inventory sourcing, payment processing, order packing, etc. Canada Post will add this time to the shipping date and quote the delivery date based on that.

- For "Type of container used to ship product" select "Unknown" if you are not sure. If shipping single product Canada post will find smallest box available and if shipping multiple products it will "find the most cost-effective box combination. http://www.canadapost.ca/business/offerings/sell_online_shipping_module/can/demo_5-e.asp

3.6 Analytics

One very important capability that's needed in order to run any kind of eCommerce solution effectively is quality analytic capabilities. Satchmo does not have a set of its own analytical capabilities, but it does have support for using Google Analytics.

In order to use Google analytics, you must have an analytics id, which you can get for free for registering with Google. Once you get this tracking code, you must enable it in the store by modifying the Site Setting to include the google code.

After enabling Google analytics, you will need to make sure that your analytics account has the "Yes, an Ecommerce website" is selected in the analytics site's setting. Additionally, you can select the "Do track site search" in order to view what people are searching for. Enter "keywords" in the query parameters search field.

Now, all of your pages will be tracked via google analytics. Satchmo also includes the additional ecommerce tracking codes so that purchases from your store can be tracked via analytics.

The google analytics code is implemented using 2 template tags. By default, they are included in the base.html and checkout/success.html pages. If you wish to alter your templates, use {% show_tracker is_secure %} to implement the tracking code and {% show_receipt %} to implement the purchase tracking capabilities. The show_tracker code must be within the <body> tags and must be before the show_receipt tags.

One final note on the google analytics capability. The actual code that is inserted in your web page is controlled by the two templates in the google-analytics templates directory. If you would like to host the google analytics javascript file on your own server (make sure you know what this means for keeping it updated), you can just modify these templates.

3.7 Payment Modules

Satchmo currently has support for several different payment modules. This document will discuss some of the particular configuration items to keep in mind for each module.

There are 3 primary types of payment modules in Satchmo:

1. Modules based on views not asking for credit card number on the merchant site:

     Google Checkout, PayPal, Sermepa.

2. Modules that collect and process a card number:

     Authorize.net Cybersource.net, Sage Pay, TrustCommerce.

3. Modules which are not typical payment gateways:

     Auto success, COD (cash on delivery), dummy, gift certificate, purchase order.

3.7.1 Authorize.net

The authorize.net module requires that you have a valid authorize.net account. In order to get your transaction key and transaction login, go to the site and complete the account registration process. When you have completed it, you can fill in the appropriate fields in the admin settings to enable this payment processor.

If you have a test account with authorize.net and you log in through https://test.authorize.net/gateway/transact.dll, then you should use the default test URL. If you do not have a test account you will get an Error 13 message unless you change the URL to https://secure.authorize.net/gateway/transact.dll. You will also need to login in to authorize.net and make sure your account has test mode turned on.

3.7.2 TrustCommerce

TrustCommerce configuration is very similar to authorize.net. You will need to go to TrustCommerce and get your account setup.

One special note with TrustCommerce - you must have the tclink libraries installed. You can get these from here.

Once the library is installed, you can fill in your login and password information in the admin config.

3.7.3 CyberSource

CyberSource operates similarly to TrustCommerce and Authorize.net. If you register at the CyberSource Site you will get all the credentials you need to configure this module.

3.7.4 PayPal

Satchmo uses Paypal's Instant Payment Notification service. The notes below describe how to configure it for your setup.

Configuration

PAYMENT_PAYPAL.**BUSINESS**
     The email address for your paypal account.

PAYMENT_PAYPAL.**BUSINESS_TEST**
     The email address for testing your paypal account.

PAYMENT_PAYPAL.**CURRENCY_CODE**
     Currency code for Paypal transactions.

     Defaults to USD.

PAYMENT_PAYPAL.**LIVE**
>   Whether to accept real payments.
>
>   Defaults to False.

PAYMENT_PAYPAL.**POST_URL**
>   The Paypal URL for real transaction posting.
>
>   Defaults to *https://www.paypal.com/cgi-bin/webscr*.

PAYMENT_PAYPAL.**RETURN_ADDRESS**
>   Where Paypal will return the customer after the purchase is complete. This can be a named url and defaults to
>   'satchmo_checkout-success'.

PAYMENT_PAYPAL.**POST_TEST_URL**
>   The Paypal URL for test transaction posting.
>
>   Defaults to *https://www.sandbox.paypal.com/cgi-bin/webscr*.

PAYMENT_PAYPAL.**URL_BASE**
>   Defaults to *^paypal/*.

Step by Step Instructions for Configuring Paypal IPN

The following steps describe how to configure Paypal IPN for production use.

1. Make sure that 'payment.modules.paypal' is listed in your INSTALLED_APPS tuple, within your settings.py file.

Then go into your /admin settings and ensure the configuration is correct for yuour site:

2. In Site Settings -> Payment Settings, Check "Accept real payments".

3. In Site Settings -> Paypal Payment Settings, enter your Merchant Paypal account email

4. In Site Settings -> Paypal Payment Settings verify that POST URL is: https://www.paypal.com/cgi-bin/webscr

5. If you want paypal to adjust your stock levels, make sure your Track inventory levels is checked

Now, configure your Paypal Merchant Account through the Paypal website:

6. In History select IPN History

7. Select the enable IPN option

8. Use the correct Notification URL. For example: http://www.mydomain.com/checkout/paypal/ipn/

Double check your Satchmo sites domain name to make sure it matches the notification url above:

9. Go to /admin/sites/site/ and make sure your domain name is the same as the one you configured about in PayPal. In the example above, under Sites put in mydomain.com

There is one note to remember about using paypal with utf-8 characters. By default Paypal will send an IPN call encoded as cp-1252 which will break when the payment module calls PayPal again in utf-8 and there are special characters in the call. The solution is to login to PayPal and change the website encoding of your account on PayPal to utf-8 in your profile settings (bottom right link on the profile page).

3.7.5 Payflow pro payment method

This payment communicates with PayPal's PayflowPro system.

You will need a payflow gateway account, and a merchant bank account to link it to. At the moment of this writing, the service was available in the United States, Australia, Canada and New Zealand only.

Dependencies

You must install python-payflowpro . It's recommended that you install it using pip:

```
pip install git+git://github.com/bkeating/python-payfowpro.git#egg=python-payflowpro'. If this doesn'
pip install -e git+git://github.com/bkeating/python-payflowpro.git#egg=python-payflowpro', but I real
```

Installation

Simply add 'payment.modules.payflowpro' to your INSTALLED_APPS

Settings

No specific django settings for this app.

Livesettings

When you activate this app, you will see a new PayflowPro Payment Settings tab in your site's livesettings page. Here you can select whether to use real payments or not, which types of credit cards you want to accept, whether to use a capture or authorize-capture sales model, the name the user will see in the checkout screens, and the URL this module will work under.

Most important are the Partner, Vendor, User, and Password settings. You will receive this data from PayPal when you sign up for your account.

What Works

Basic authorize and capture work.

TODO

- write unit tests.
- test authorize + release (when you cancel an order) workflow
- write support for recurrent payments
- sort the livesettings options

3.7.6 Google Checkout

There are a couple of different ways to integrate with Google Checkout. Satchmo uses the XML option. When configuring your checkout account via Google, select: "Option A - Configure your form to submit directly to Google Checkout."

It is recommended that you review the Google checkout documentation in order to understand how the process works.

3.7.7 Sage Pay (Formerly Protx)

This processor is included in the default Satchmo store.

3.7.8 Sermepa

This processor is included in the default Satchmo store.

3.8 Email Verification

By default, Satchmo doesn't ask users to verify their email addresses. If you'd like to, you can configure Satchmo to use django-registration.

1. Download and extract django-registration. Make sure that the registration folder is on your pythonpath.

2. Edit local_settings.py:

    - Set ACCOUNT_ACTIVATION_DAYS to an integer value. This setting determines how long it takes for an activation code to expire.

3. Edit settings.py:

    - Add 'registration' to INSTALLED_APPS.

4. Run "python manage.py syncdb".

5. In Site Administration -> Site Settings:

    - Account Verification -> Email

    - Days to verify Account -> Your integer value here

Modifying A Store

4.1 Store Customization

The real power of Satchmo is not in what it does out of the box but what the framework allows you to do with minimal modification. As you start to develop your online presence you will have a whole range of different ideas on how you would like to make your store look and feel. Satchmo provides a simple store layout that you can modify as much or as little as you wish. There are several ways to tweak Satchmo to your needs. The following list is in order of the relative ease of customization:

1. CSS customization

2. Template customization

3. URL customization

4. Checkout process customization

5. Shipping module customization

6. Payment module customization

7. Changing views

8. Creating custom template tags

4.1.1 CSS Customization

The simplest way to modify Satchmo is to change the attributes in the css file used by the default shop. The current style.css file is located in /satchmo/static/css. You can modify this file directly (not recommended) or create a new CSS file and modify the base.html template file to point to your new and improved css file.

The benefits of this approach is that it is simple to implement and makes it easy to update Satchmo as changes are made. The downside is that there is only so much you can do with css changes.

4.1.2 Template Customization

Template customization allows you to completely alter the look and feel of your site without having to resort to coding. The majority of the display elements are accessible via css or template changes. Before making any template changes, it is useful to understand Django templates in general. The Django Template Author Guide is the best place to start.

The other very important concept to understand is how to setup your environment so that you can selectively modify the templates that you need to. You should not modify the templates directly in the Satchmo distribution. Doing so will make it more difficult to upgrade and maintain your Satchmo installation.

In the default Satchmo settings.py, our template loaders are:

```
TEMPLATE_LOADERS = (
    'django.template.loaders.filesystem.load_template_source',
    'django.template.loaders.app_directories.load_template_source',
    )
```

These template loaders look for the templates in two different ways. The first one, the filesystem loader, looks for the templates in the directories you tell it in your settings_local.py file:

```
TEMPLATE_DIRS = ('/path/to/your/templates',)
```

The second loader, the app_directories loader, looks for the templates in a directory named "templates" in each application directory.

Any time the system tries to find a template, it follows the order given above. First it asks the filesystem loader to look in the directories it has been given. If it finds the template there, it returns that one. Next it asks the app_directories loader to look in the app directories for that template. These are the "fallback" templates to use.

For example, if you wanted to override the default category page template, you can determine that it lives at `satchmo/apps/product/templates/product/category.html` It will be found by the app_directory loader if the filesystem loader doesn't find it.

Knowing that, you can tell that the proper place to put your overridden template is at: `/path/to/your/templates/product/category.html`

In other word, you just chop the directory tree off at the word "templates" and build your override tree that way.

Another example might be useful. If you wish to override the wishlist index page, which usually lives at: `satchmo/apps/satchmo_ext/wishlist/templates/index.html`, you would make a directory `wishlist/templates` and put an index.html file in it.

Now, make the changes to the templates that you have to. Don't forget, this is also the place where you can change the email messages sent to users as well as the pdf documents created by the admin interface. The amount of power and flexibility here is much higher than just about any "shopping cart" application out there!

4.1.3 URL Customization

Another nifty feature of Django is the flexibility in configuring the URLs of an application. Because a web store frequently requires good search engine ranking, you will most likely want to do everything in your power to help your store rise in the rankings. Using good, clean descriptive URLs is a huge plus. Another benefit of flexible URLs is that allows you to easily integrate Satchmo with an existing site or differentiate your site with a creative scheme (or language).

For convenience, urls can be modified by changing the SHOP_URLS variable in the local_settings.py file. Any url assigned to this variable will override the existing url naming scheme.

Additionally, the Satchmo settings application allows you to change some of the names of urls associated with products and categories.

4.1.4 Shipping Module Customization

The comments in the dummy.py shipping module walk through how to configure shipping for your unique needs.

4.1.5 Payment Module Customization

See *Custom Payment Modules*

4.1.6 Using Signals

See *Signals in Satchmo* for an explanation of the available signals and how to use them to customize your store.

4.1.7 Changing Views

In some instances, you may wish to selectively override specific Satchmo views. Satchmo includes a useful utility-`replace_urlpattern` in satchmo_utils.urlhelper to exchange stock urls with your own.

Here is a simple example of using it to replace the quick_order view. In this example, we wish to change the items that are displayed on the quick order page so that only featured items are shown. The full urls.py is shown below:

```python
from django.conf.urls import *

# Uncomment the next two lines to enable the admin:
# from django.contrib import admin
# admin.autodiscover()

from satchmo_store.urls import urlpatterns

from satchmo_utils.urlhelper import replace_urlpattern
from product.models import Product

product_list = Product.objects.filter(featured=True)

replacement = url(r'^quickorder/$', 'satchmo_store.shop.views.cart.add_multiple',
            {'products': product_list}, 'satchmo_quick_order')
replace_urlpattern(urlpatterns, replacement)
```

4.1.8 Custom Template Tags and Filters

If you find yourself in a situation where you need to display additional information on the page or where you need to change the way data is displayed, you should consider writing a custom template tag or filter. The general process for using the tag or filter would be:

1. Create the new tag in your application's templatetags directory

2. Selectively override the Satchmo templates that use the tag

3. Import the new tag at the top of each template and use it in the template

Example tags

Here is a quick example showing how you can create an inclusion tag to display a list of new arrivals or featured items.

First, make sure you have a `templatetags` directory in your `localsite` directory (this assumes you are using the default satchmo setup). In this directory, create a file called `local_tags.py`

Your `local_tags.py` should look like this:

```python
from django import template
from product.models import Product
register = template.Library()

@register.inclusion_tag('localsite/new_arrivals.html')
def show_new_arrivals(number):
```

```
    new_arrivals = Product.objects.all().order_by('date_added')[:number]
    return {'new_arrivals': new_arrivals}


@register.inclusion_tag('localsite/featured_items.html')
def show_featured_items(number):
    featured_items = Product.objects.filter(featured=True)[:number]
    return {'featured_items': featured_items}
```

Each of these tags also will need a custom html template that will be used to render the items. Inside your template directory, create the following files /localsite/new_arrivals.html:

```
{% for product in new_arrivals %}
    <ul><a href="{{ product.get_absolute_url }}">{{product.translated_name}}</a></ul>
{% endfor %}
```

As well as this file /localsite/featured_items.html:

```
{% for product in featured_items %}
    <ul><a href="{{ product.get_absolute_url }}">{{product.translated_name}}</a></ul>
{% endfor %}
```

In order to use these tags, add them to your templates:

```
{% extends "shop/base.html" %}
{% load local_tags %}

<h1>New Arrivals</h1>
    <ul>
    {% show_new_arrivals 5 %}
    </ul>
<h1>Featured Items</h1>
    <ul>
    {% show_featured_items 5 %}
    </ul>
```

This feature is a very powerful and simple way to add custom information into your custom satchmo site.

4.2 Customizing Admin

Satchmo does not make extensive modifications to the default Django admin. However, it is relatively straightforward to make changes to the admin for your needs - without modifying Satchmo's core.

4.2.1 Modifying the Product Model

The default Product admin model displays all possible fields and makes some assumptions about what you would like to see. If you would like to modify the admin to use special widgets or change fields that are displayed, use admin.site.unregister to unregister the current product model and add your own.

Assuming you have a localapp admin.py file, use code similar to the example below:

```
from django.contrib import admin
from django.db import models
from product.models import Product
from product.admin import ProductOptions
from tinymce.widgets import AdminTinyMCE
```

```
"""
Rest of your admin models go here.
"""

admin.site.unregister(Product)

class CustomProductAdmin(ProductOptions):
    formfield_overrides = {
        models.TextField: {'widget': AdminTinyMCE},
        }
    list_display = ('name', 'unit_price', 'items_in_stock', 'active','featured')
    list_display_links = ('name',)

admin.site.register(Product, CustomProductAdmin)
```

This code does a couple of things:

- Replace the default Text Editor widget with TinyMCE widget

- Changes the columns in the list_display to be a subset of the default

- Ensures that the name is a link to the product

Note: These changes must be included in an application that is listed after all the other Satchmo Apps in `INSTALLED_APPS`

4.2.2 Using Mix-ins to customize the admin

Mix-ins are a very powerful feature available in python that allow for advanced customization of Satchmo. With mix-ins you can add arbitrary elements (functions, model attributes, etc...) to any class/model. The full uses of mix-ins are outside the scope of this document but let's start with an example.

Suppose you want to show product SKUs of all the products in an order on the Product Orders page. The screenshot below shows you what the current order screen looks like:

We will modify it to show all the skus in an order like this:

Here is a chunk of code that would create a list of skus for a given order:

```python
def order_sku(self):
    allsku = ''
    for item in self.orderitem_set.all():
        allsku += "%s<br />" % item.product.sku
    return allsku
order_sku.allow_tags = True
order_sku.short_description = "Order SKUs"
```

You could then add 'order_sku' to the list display of Order. The problem is, where do you put the chunk of code listed above? You could add it to the Order models but that would be forking Satchmo and later when it was time to upgrade to the next version of Satchmo things could be much more difficult. The solution is mix-ins.

Mix-ins allow you to dynamically change the inheritance of a class. We will change Orders from looking like this:

```python
class Order(models.Model):
```

to this:

```python
class Order(models.Model, ourNewClass):
```

In order to accomplish this without actually modifying existing Satchmo done, you can put order_sku in a class:

```python
class OrderExtend:
    def order_sku(self):
        allsku = ''
        for item in self.orderitem_set.all():
            allsku += "%s<br />" % item.product.sku
        return allsku
    order_sku.allow_tags = True
    order_sku.short_description = "Order SKUs"
```

and adding:

```python
Order.__bases__ += (OrderExtend,)
```

The last bit takes the __bases__ of Order (which are its base inheritance classes) and adds OrderExtend to it. Because Order now inherits from OrderExtend we can use order_sku. To make skus show up in the admin we create `localsite/admin.py` and put all of our code in it so it looks like this:

```python
from django.contrib import admin
from django.db import models
from satchmo_store.shop.models import Order, OrderItem
from satchmo_store.shop.admin import OrderOptions


class OrderExtend:
    def order_sku(self):
        allsku = ''
        for item in self.orderitem_set.all():
            allsku += "%s<br />" % item.product.sku
        return allsku
    order_sku.allow_tags = True
    order_sku.short_description = "Order SKUs"

Order.__bases__ += (OrderExtend,)

admin.site.unregister(Order)


class CustomOrderAdmin(OrderOptions):
    list_display = ('id', 'order_sku', 'contact', 'time_stamp',
                    'order_total', 'balance_forward', 'status',
                    'invoice', 'packingslip','shippinglabel')

admin.site.register(Order, CustomOrderAdmin)
```

Obviously you could do a lot more with mix-ins. You can add arbitray functions and attributes to any model and any class. A much longer explaination that is not django specific but has more examples (and the document used as a reference for writing this up) can be found here

4.3 Custom Payment Modules

While Satchmo currently has support for several different payment modules, you may have unique needs or the desire to create your own payment processor module. This document will discuss how to create your own payment modules. If you do decide to create your own module, please let us know so we can include it back into the satchmo core and make the framework that much more robust.

4.3.1 Overview

Satchmo's payment processor modules are meant to be as modular as possible. For many types of payment processors, you should be able to use one of the existing modules as a basis for creating your own.

All of the modules are stored in the /payment/modules directory. If you take a quick look at any one of the subdirectories, you will see a number of files:

- __init__.py

- config.py

- processor.py

- urls.py

- views.py

The __init__.py is required so that satchmo can import the files. There is no need to put any code in this file. Just be sure it exists!

The rest of the files are described below.

4.3.2 Building your processor

The `processor.py` file is where the majority of the heavy lifting is done. The processor does 4 things:

- Sets up its configuration for the service (`__init__`)

- Takes order data and formats it in the appropriate manner (`prepareData`)

- Sends the data to the processing server/url (`authorize_payment`, `capture_payment`, `capture_authorized_payment`) and returns the results.

Of them, only `capture_payment` is required. If the processor can both authorize and process, then you need to add a function `can_authorize`, which returns True, and implement the other two methods. See the `dummy` module for an example.

Optionally, the processor can include include test code so that it is easy to verify from the command line.

Here is a stub you can use to create your own processor:

```
from payment.modules.base import BasePaymentProcessor, ProcessorResult

class PaymentProcessor(BasePaymentProcessor):
    def __init__(self, settings):
        # Set up your configuration for items like
        # Test server, url, various flags and settings
        super(PaymentProcessor, self).__init__('example', settings)

    def capture_payment(self):
        # Send the data via the appropriate manner and return a ProcessorResult
        # object.
```

Refer to the dummy, authorize.net, cybersource or trustcommerce modules for various examples to help you through the process.

4.3.3 Configuration

Each processor will have unique variables that need to be set. The `config.py` file is where you can leverage the Satchmo settings capability to add your unique variables. Please refer to the *Basic Configuration* in order to understand how the system works. For more examples, the existing working modules are great examples of what to setup. The basic format is:

Create the new configuration group:

```
PAYMENT_GROUP = ConfigurationGroup('PAYMENT_MYNEWPROCESSOR',
    _('My New Processor Payment Settings'),
    ordering=102)
```

Note: The key of the `ConfigurationGroup` must be the module name, upper-cased.

For example, if your custom payment module was located in `mypaymentmodules.mynewprocessor`, the `ConfigurationGroup` should be given the key `PAYMENT_MYNEWPROCESSOR`. The reason for this is that the `active_gateways()` function in `payment/__init__.py` attempts to automatically determine these keys by appending the upper-cased module name to `PAYMENT_`.

Now register the settings you need:

```
config_register_list(
    BooleanValue(PAYMENT_GROUP,
        'LIVE',
        description=_("Accept real payments"),
        help_text=_("False if you want to be in test mode"),
        default=False),

    StringValue(PAYMENT_GROUP,
        'LABEL',
        description=_('English name for this group on the checkout screens'),
        default = 'Credit Cards',
        help_text = _('This will be passed to the translation utility')),

    StringValue(PAYMENT_GROUP,
        'URL_BASE',
        description=_('The url base used for constructing urlpatterns which will use this module'),
        default = r'^credit/'),

    MultipleStringValue(PAYMENT_GROUP,
        'CREDITCHOICES',
        description=_('Available credit cards'),
        choices = (
            (('Amex', 'American Express')),
            (('Visa','Visa')),
            (('Mastercard','Mastercard')),
            (('Discover','Discover'))),
        default = ('Visa', 'Mastercard', 'Discover')),

    StringValue(PAYMENT_GROUP,
        'PASSWORD',
        description=_('Your Processor password'),
        default=""),

    BooleanValue(PAYMENT_GROUP,
        'EXTRA_LOGGING',
        description=_("Verbose logs"),
        help_text=_("Add extensive logs during post."),
        default=False)
)
```

All of these settings can be accessed in your __init__ method (shown above). For example, the LIVE value above can be accessed by using settings.LIVE.value

4.3.4 Views

Most payment processing have similar steps:

- Collect demographic information

- Collect payment information

- Confirm info is correct

- Return a status

The views.py file contains the information that maps your processor views to the existing views or your own custom view.

For most people, the views contained in payment.views will be sufficient. The example below maps these views to views already available in Satchmo:

```
from livesettings import config_get_group
from payment.views import confirm, payship


def pay_ship_info(request):
    return payship.credit_pay_ship_info(request, config_get_group('PAYMENT_MYNEWPROCESSOR'))


def confirm_info(request):
    return confirm.credit_confirm_info(request, config_get_group('PAYMENT_MYNEWPROCESSOR'))
```

However, there is nothing stopping you from creating your own view:

```
def confirm_info(request):
    # Do a lot of custom stuff here
    return render_to_response(template, context)
```

All of the current satchmo payment views are in `/payment/views`. Please review these before trying to build one of your own!

4.3.5 Url configuration

Now that you have built your processor, configured your settings and built your views, you need to tell Satchmo how to access these views. This is where the `urls.py` file is useful.

For most processors, a simple file would look like this:

```
from django.conf.urls import *
from livesettings import config_value, config_get_group


config = config_get_group('PAYMENT_MYNEWPROCESSOR')

urlpatterns = patterns('satchmo',
    (r'^$', 'mypaymentmodules.myprocessor.views.pay_ship_info',
                {'SSL':config.SSL.value}, 'MYNEWPROCESSOR_satchmo_checkout-step2'),
    (r'^confirm/$', 'payment.modules.trustcommerce.views.confirm_info',
                {'SSL':config.SSL.value}, 'MYNEWPROCESSOR_satchmo_checkout-step3'),
    (r'^success/$', 'payment.views.checkout.success',
                {'SSL':config.SSL.value}, 'MYNEWPROCESSOR_satchmo_checkout-success'),
)
```

The nice thing about this file is that it allows you to easily plug in the views you need and rename the urls to whatever form you need. Just make sure to maintain the naming convention for the urls as shown above.

4.3.6 Enabling the new module

In order to enable your new payment processor, you must add it to your INSTALLED_APPS setting in your settings.py. For example:

```
INSTALLED_APPS = (
        ...
        'payment',
        'mypaymentmodules.myprocessor',
        ...
```

4.3.7 Conclusion

Hopefully this document will help you get started in creating your own payment modules. Before trying to tackle one on your own, take some time to look at the existing models and get a feel for how things have been done. Once you are comfortable, we suggest copying one of the modules and using it as a starting point for your subsequent efforts. If you get stuck, please feel free to ask the mailing list for help.

4.4 Custom Product Attributes

Satchmo's product models allow you to capture and present much of the core information you need to sell a product. However, there are instances where you might want to present additional information tailored to your site. If you think of the example of selling books online, you might want all of your products to display something like:

- Pages

- Publisher

- ISBN

- Author

You can certainly put this information into the description field but in many cases you will want these to be separate fields and enforce some validation on the data that can be entered. Attribute Options can do this for you.

4.4.1 Creating Attribute Options

New in version 0.9.1.

New versions of Satchmo (> 0.9) include the ability to define options, validations for these options and custom error messages. To use these, you will first need to define some Attribute Options in the Admin.

See this screenshot for some examples:

As you can see, there are multiple attributes defined that we can add to our products. Looking at the "Book Pages" shows you some of the rich options you have to configure this behavior:

Description Information displayed to the user on the product page to describe the attribute

Attribute Name Slug used for data storage

Field Validations List of validations available to perform on the data. Satchmo has some included but you may create your own too.

Sort Order Control the order the options are displayed

Error Message If the validation fails, this message will be displayed to the user.

4.4.2 Using Attributes on Products

Now that we have an attribute for pages in a book, we can add this to our Robot Attacks book.

Select the "Robot Attack!" book from the product list and scroll down to the Product Attributes section:

If you enter an invalid number, you will get your custom error message. If you enter a valid number, you can view it on your page now:

4.4.3 Customizing Attribute Options

This feature is very useful but you can go even farther in your customization by creating your own Field Validations.

If you would like to create a custom validation to make sure the site admin enters a 3 digit integer, create a simple validation function like this:

```python
def validation_3digits(value, product=None):
    """
    Validates that value is a 3 digit integer number.
    No change is made to the value
    """
    try:
        check = int(value)
        if len(value) == 3:
            return True, value
        else:
            return False, value
```

```
    except:
        return False, value
```

Assuming you have placed this function in the utils.py file inside your localsite app. You can add it to your settings.py (or local_settings.py) file:

```
SATCHMO_SETTINGS = {
    'SHOP_BASE' : '',
    'MULTISHOP' : False,
    'ATTRIBUTE_VALIDATIONS': [('localsite.utils.validation_3digits','3 Digits'),]

}
```

Now, when you choose to add an Attribute Option, you can use your 3 digit validator to make sure that the site admin enters the data you need.

4.4.4 Additional Capabilities

The example above is fairly simple. As you will notice, the current product is passed to the validation function. You can use this to do more complex lookups. Additional, the validation function can modify the value that is passed to it. For instance, you may wish to make sure the value is capitalized or structured in a consistent date format.

4.5 Custom Product Modules

Though Satchmo includes a number of versatile product types, many projects have needs that are best met by creating a custom product module. A custom product module is a module with a product model which deviates from the normal behavior of Satchmo.

To work with custom product modules, you will need a basic understanding of Django apps, projects, and templates. You should understand the default behavior of Satchmo before trying to customize it.

4.5.1 Building your module

In a custom product module, most of the work is done in the `models.py` and template files. You will also need an `admin.py` file like for any other model which shall be available in the admin. This section contains several example files that you can add to a new or existing app and extend.

A basic `models.py` file with a custom product model looks like this:

```python
from django.db import models
from django.utils.translation import ugettext_lazy as _
from product.models import Product

SATCHMO_PRODUCT=True


def get_product_types():
    """
    Returns a tuple of all product subtypes this app adds
    """
    return ('MyNewProduct', )


class MyNewProduct(models.Model):
    product = models.OneToOneField(Product, verbose_name=_('Product'),
        primary_key=True)
```

```
    def _get_subtype(self):
        """
        Has to return the name of the product subtype
        """
        return 'MyNewProduct'

    def __unicode__(self):
        return u"MyNewProduct: %s" % self.product.name

    class Meta:
        verbose_name = _('My New Product')
        verbose_name_plural = _('My New Products')
```

The important parts are the attribute `MyNewProduct.product` which links to the related `product.models.Product` and the method `MyNewProduct._get_subtype()` which returns a string representation of this model.

The function `get_product_types()` returns a list of all custom product models this app adds. This method is used to discover the custom product models from all installed apps.

This is the corresponding `admin.py` file. This file is needed to make your models visible in Django's admin app:

```
from django.contrib import admin
from models import MyNewProduct

admin.site.register(MyNewProduct)
```

4.5.2 Configuration

Once you've created the above files in your app, you have all the code necessary to use your new product model in Satchmo. All that's left is the configuration.

1. Make sure that the app with your product model is in your project's `INSTALLED_APPS` setting.

2. Run `python manage.py syncdb` in your project directory.

You can now use the new product model in the same way that you would use one of Satchmo's default product types. You will find an "Add MyNewProduct" link in the "Product Subtypes" section of each product's admin page.

4.5.3 Extending the model and templates

A product model is a Django model with a `OneToOneField` to `satchmo.product.models.Product` and a `_get_subtype` method. You may add new fields and behavior as you would with any other Django model.

When Satchmo renders a product page, it looks for a template named `product/detail_productname.html` (in all lowercase). If the template is not found, Satchmo uses the `base_product.html` template.

As an example, say you are using `MyNewProduct` from the previous example and you want to extend it to display a special title on the product's page. First, you would add a `CharField` named `title` to the existing model and to the table in your database (or just drop the table and run `syncdb`). Then, create a template named `product/detail_mynewproduct.html` with the following content:

```
{% extends "product/product.html" %}

{% block title %}{{ product.mynewproduct.title }}{% endblock title %}
```

If you create a `MyNewProduct` and view its page in the store, the page will have the title you assigned it in the admin app. Notice that the `MyNewProduct` is accessed as an attribute of `product`.

For more examples, look at `product/models.py`, `templates/base_product.html`, and `templates/product/` in the Satchmo source code.

4.5.4 Using model inheritance

There is another possibility for extending the product model using model inheritance. You can find more information in the first installment of the Satchmo Diaries.

> **Warning:** Using inheritance is currently not recommended, because it has some issues when it is used without care. Some methods of `product.models.Product` currently do not expect to find subtypes which inherit attributes and methods from `product.models.Product`.

If you want to use inheritance, make sure *not* to add your model to the list returned by `get_product_types()`, else you might get some trouble with infinite recursions. An alternative would be to extend the current implementation to be able to handle models which inherit from `product.models.Product`.

So, your `models.py` file should look like this:

```python
from django.db import models
from django.utils.translation import ugettext_lazy as _
from product.models import Product

class MyNewProduct(Product):

    # TODO: your attributes here

    objects = ProductManager()

    def __unicode__(self):
        return u"MyNewProduct: %s" % self.name

    class Meta:
        verbose_name = _('My New Product')
        verbose_name_plural = _('My New Products')
```

To make this model available in the admin interface you should create a corresponding `admin.py`:

```python
from django.contrib import admin
from models import MyNewProduct
from product.admin import ProductOptions

# TODO: do your customizations here
class MyNewProductAdmin(ProductOptions):
    pass

admin.site.register(MyNewProduct, MyNewProductAdmin)
```

4.5.5 Conclusion

This document should get you started in customizing Satchmo through the product model. If you need help with something discussed here or with more advanced topics, feel free to ask the mailing list.

Finally, if you create a product model that others may find useful, please consider contributing it to the Satchmo community.

4.6 Translating Content

When translating Satchmo to your desired language, you will generally need to translate the words in the templates as well as the text used in your product and category descriptions. The Django Internationalization documentation is an excellent resource on the overall capability available through Django. The sections below describe the Satchmo specific steps.

4.6.1 Using Transifex

Satchmo is available on transifex which describes itself as "... an open service allowing people to collaboratively translate software, documentation and other types of projects."

This application provides an easy to use web interface for translating Satchmo and submitting the translations back to Satchmo so they may be incorporated into the application.

The specific Satchmo transifex site is here.

If you prefer to use your own tools for translation, the sections below describe the steps to generate and maintain your translations.

4.6.2 Template Translations

1. Change to the directory where your Satchmo source is stored (somewhere on your PYTHONPATH). You can run the following find command to compile all of the langauge files for the satchmo apps:

```
find . -name locale -exec sh -c 'cd $0 && cd ../ &&
        /path/to/django-trunk/django/bin/django-admin.py makemessages -l de -e html,txt,rml' {} \;
```

In the example above, replace "de" with the language code for the message file you want to create. The language code, in this case, is in locale format. For example, it's pt_BR for Brazilian Portuguese and de_AT for Austrian German. The first two letters of the locale is a language code defined in ISO 639-1 . The second and optional part consists of an underscore followed by a two letter country code as defined in ISO 3166-1 . For the command above to work you need to have python and xgettext installed.

This will extract strings that need to be translated from the Satchmo code and templates. Upon completion of this command these strings along with references to where they are used are stored in multiple *locale/de/LC_MESSAGES/django.po* (again, replace "de" with your own locale).

If the *django.po* file already exists the above command will update it and add new strings if any are discovered.

2. Translate the *django.po* file of your locale using a translation editor:

   • poedit

   • kbabel

   • gtranslator

3. After translating the file, you must compile it to an .mo file. To do this, you must make sure that your local_settings.py file has a LOCALE_PATHS variable defined (it can be a blank string).

4. Compile the file using the compilemessages command (run from your satchmo source directory):

```
find . -name locale -exec sh -c 'cd $0 && cd ../ &&
                          /path/to/django-admin.py compilemessages' {} \;
```

5. (Optional but encouraged) Submit a ticket with your translation.

4.6.3 Making a translation available

Django will choose a translation based on the user's browser settings, but this is not always the solution you may be looking for. To allow users to choose a translation using Satchmo's language selection form, you must add the list of available translations to your settings.py (or local_settings.py):

```
LANGUAGES = (
            ('en', "English"),
            ('fr', "Français"),
            ('de',"Deutsch"),
            ('es', "Español"),
            ('he',""),
            ('it',"Italiano"),
            ('ko', ""),
            ('sv', "Svenska"),
            ('pt-br',"Português"),
            ('bg',""),
            ('tr',"Türkçe"),
            )
```

Then, edit your settings.py file to enable the i18n urls.:

```
#### Satchmo unique variables ####
from django.conf.urls import patterns, include
SATCHMO_SETTINGS = {
    'SHOP_BASE' : '',
    'MULTISHOP' : False,
    'SHOP_URLS' : patterns('', (r'^i18n/', include('l10n.urls')),)
}
```

Also, make sure to add 'django.core.context_processors.i18n' to TEMPLATE_CONTEXT_PROCESSORS so that templates will use the correct language code.

The default store setup will show the form to allow the store user to select their translation from the above list. If you wish to disable this option, disable `allow_translation_choice` by follow the instructions here *Changing the L10N Settings*.

4.6.4 Translating dependencies

Unless a translation to your language is already available, you may also need to translate packages on which Satchmo depends:

- Django
- Django Registration

Such translations are to be submitted to the corresponding project and not to the Satchmo project.

4.6.5 Content Translations

After you have translated the templates, you will need to add your translations for the product information. Satchmo has extensive support for multiple languages in your product and category descriptions.

After you have created a product, you can use the Product Translation section to create translated Name, Short Description and Full Description of the product. Creation is straightforward and allows you to version your changes so that you can try different translations to see which is most effective.

4.7 Satchmo Templates

One of the powerful options available to all Satchmo stores is the availability of the robust Django templating language to control all of the visual elements of this store. There are over 140 templates used within Satchmo. This section of the document describes where they are and the high level overview of what they are used for. Until more details are included in this section, the best way to learn more about these templates is to browse through and review the ones you are interested in. If you have a particular view that you would like to customize and are unsure which templates are being rendered, then the django_debug_toolbar can help you out. Go ahead and install it and use it for debugging and troubleshooting your site.

It is also important that you review and understand how to setup your templates so that they are easy to modify and maintain. Please review *Template Customization* before modifying templates.

4.7.1 Templates by View

See *Satchmo Views*.

4.7.2 Templates by App

Payment

These templates control some of the payment specific display elements.

`payment/templates/payment/admin/charge_remaining_confirm.html`
    Used in the admin to support paying any remaining balance on an order.

`payment/templates/payment/_order_payment_summary.html`
    Used in the checkout process to show a summary of all the payments that have been applied to an order.

Payment/modules/

Templates in this directory are used to modify the behavior of the giftcertificate payment process. They are useful for understanding how to further modify the checkout process.

`giftcertificate/templates/shop/checkout/giftcertificate/pay_ship.html`

`giftcertificate/templates/shop/checkout/giftcertificate/confirm.html`

`giftcertificate/templates/shop/checkout/confirm.html`

`giftcertificate/templates/giftcertificate/balance.html`

`giftcertificate/templates/giftcertificate/_order_summary.html`

`giftcertificate/templates/product/detail_giftcertificateproduct.html`

`giftcertificate/templates/plugins/order_details.html`

L10n

L10n is used for localizing Satchmo.

`l10n/templates/l10n/_language_selection_form.html`
> Displays the language selection form that can be enabled through the admin.

Satchmo_ext

Satchmo externals contains several useful applications that you may optionally enable for your store.

wishlist

`wishlist/templates/wishlist/login_required.html`

`wishlist/templates/wishlist/plugins/product_add_buttons.html`

`wishlist/templates/wishlist/plugins/shop_sidebar_actions.html`

`wishlist/templates/wishlist/index.html`

upsell

`upsell/templates/upsell/plugins/product_form.html`

`upsell/templates/upsell/product_upsell.html`

recentlist

`recentlist/templates/recentlist/_recently_viewed.html`

`recentlist/templates/recentlist/plugins/shop_sidebar_primary.html`

productratings

`satchmo_ext/productratings/templates/productratings/_product_ratings.html`

`satchmo_ext/productratings/templates/productratings/plugins/sidebar_links.html`

`satchmo_ext/productratings/templates/productratings/plugins/product_footer.html`

`satchmo_ext/productratings/templates/productratings/product_rating_form.html`

`satchmo_ext/productratings/templates/productratings/_render_product_ratings.html`

productfeeds

`satchmo_ext/product_feeds/templates/product_feeds/plugins/admin_tools.html`

newsletter

`satchmo_ext/newsletter/templates/newsletter/update_form.html`

`satchmo_ext/newsletter/templates/newsletter/update_results.html`

`satchmo_ext/newsletter/templates/newsletter/unsubscribe_form.html`

`satchmo_ext/newsletter/templates/newsletter/ajah.html`

`satchmo_ext/newsletter/templates/newsletter/subscribe_form.html`

brand

`satchmo_ext/brand/templates/brand/view_brand.html`

`satchmo_ext/brand/templates/brand/index.html`

Product

The product templates are used to display all aspects of products in your store. The admin templates are used to manage various portions of Satchmo's custom admin interface.

`product/templates/product/detail_configurableproduct.html`

`product/templates/product/category_index.html`

`product/templates/product/minimum_order.html`

`product/templates/product/cart_detail_customproduct.html`

`product/templates/product/detail_customproduct.html`

`product/templates/product/product.html`

`product/templates/product/best_sellers.html`

`product/templates/product/category.html`

`product/templates/product/recently_added.html`

`product/templates/product/some_discount_eligible.html`

`product/templates/product/product_discount_eligible.html`

`product/templates/product/cart_detail_subscriptionproduct.html`

`product/templates/product/sale_details.html`

`product/templates/product/detail_subscriptionproduct.html`

`product/templates/product/best_ratings.html`

admin

`product/templates/product/admin/inventory_form.html`

`product/templates/product/admin/product_export_form.html`

`product/templates/product/admin/variation_manager.html`

`product/templates/product/admin/product_import_result.html`

product/templates/product/admin/variation_manager_list.html

product/templates/admin/product/productvariation/change_form.html

product/templates/admin/product/product/change_form.html

product/templates/admin/product/configurableproduct/change_form.html

product/templates/admin/product/change_form.html

Keyedcache

keyedcache/templates/keyedcache/view.html

keyedcache/templates/keyedcache/stats.html

keyedcache/templates/keyedcache/delete.html

Satchmo_Store

satchmo_store/shop/templates/404.html

satchmo_store/shop/templates/500.html

satchmo_store/shop/templates/base.html

satchmo_store/shop/templates/shop/json.html

satchmo_store/shop/templates/shop/order_tracking.html

satchmo_store/shop/templates/shop/google-analytics/receipt.html

satchmo_store/shop/templates/shop/google-analytics/signup.html

satchmo_store/shop/templates/shop/google-analytics/tracker.html

satchmo_store/shop/templates/shop/google-analytics/adwords_conversion.html

satchmo_store/shop/templates/shop/_messages.html

satchmo_store/shop/templates/shop/contact_form.html

satchmo_store/shop/templates/shop/404.html

satchmo_store/shop/templates/shop/contact_thanks.html

satchmo_store/shop/templates/shop/base.html

satchmo_store/shop/templates/shop/order_history.html

satchmo_store/shop/templates/shop/admin/_orderpayment_list.html

satchmo_store/shop/templates/shop/admin/_ordercount_list.html

satchmo_store/shop/templates/shop/admin/order_sidebar.html

satchmo_store/shop/templates/shop/admin/_customorder_management.html

satchmo_store/shop/templates/shop/_search.html

satchmo_store/shop/templates/shop/download.html

satchmo_store/shop/templates/shop/_order_details.html

satchmo_store/shop/templates/shop/email/contact_us.txt

satchmo_store/shop/templates/shop/email/order_complete.txt|html
    This template is used to render the order receipt email to the customer. If html email sending is configured and there is a corresponding ".html" file, an html formatted email will be sent to the customer.

satchmo_store/shop/templates/shop/email/order_placed_notice.txt|html
    This template is used to render the store owner notification that an order has been placed. If html email sending is configured and there is a corresponding ".html" file, an html formatted email will be sent to the store owner.

satchmo_store/shop/templates/shop/email/order_shipped.txt

Note: For all of the email templates, if there is a corresponding .html template and the admin setting "Send HTML Email" is enabled, an html formatted email will be sent to the recipient.

satchmo_store/shop/templates/shop/_order_tracking_details.html
    This template can be modified to include additional information about the completed purchase. For instance, if you have a downloadable product and would like to provide a link to it when the purchase is completed, modify the template with this code:

```
{% if order.downloadlink_set.all %}
<h4>Download now</h4>
{% for dl_link in order.downloadlink_set.all %}
<a href="{{ dl_link.get_absolute_url }}">{{ dl_link.product_name }}</a><br/>
{% endfor %}
{% endif %}
```

satchmo_store/shop/templates/shop/_jquery.html
    The path to jquery is included here. If you choose to host jquery somewhere else or use a different version, override this template.

satchmo_store/shop/templates/shop/_jquery_form.html
    Similar to the _jquery.html template, this template holds the location of the jquery form library. Override this template to point to a new location.

satchmo_store/shop/templates/shop/multiple_product_form.html

satchmo_store/shop/templates/shop/_blackbird_logging.html

satchmo_store/shop/templates/shop/checkout/failure.html

satchmo_store/shop/templates/shop/checkout/form.html

satchmo_store/shop/templates/shop/checkout/purchaseorder/pay_ship.html

satchmo_store/shop/templates/shop/checkout/purchaseorder/confirm.html

satchmo_store/shop/templates/shop/checkout/base_confirm.html

satchmo_store/shop/templates/shop/checkout/authentication_required.html

satchmo_store/shop/templates/shop/checkout/success.html

satchmo_store/shop/templates/shop/checkout/cod/pay_ship.html

satchmo_store/shop/templates/shop/checkout/cod/confirm.html

satchmo_store/shop/templates/shop/checkout/paypal/pay_ship.html

satchmo_store/shop/templates/shop/checkout/paypal/confirm.html

satchmo_store/shop/templates/shop/checkout/protx/pay_ship.html

satchmo_store/shop/templates/shop/checkout/protx/confirm.html

satchmo_store/shop/templates/shop/checkout/pay_ship.html

satchmo_store/shop/templates/shop/checkout/sermepa/pay_ship.html

satchmo_store/shop/templates/shop/checkout/sermepa/confirm.html

satchmo_store/shop/templates/shop/checkout/google/pay_ship.html

satchmo_store/shop/templates/shop/checkout/google/confirm.html

satchmo_store/shop/templates/shop/checkout/balance_remaining.html

satchmo_store/shop/templates/shop/checkout/empty_cart.html

satchmo_store/shop/templates/shop/checkout/confirm.html

satchmo_store/shop/templates/shop/checkout/base_pay_ship.html

satchmo_store/shop/templates/shop/checkout/auto/pay_ship.html

satchmo_store/shop/templates/shop/search.html

satchmo_store/shop/templates/shop/index.html

satchmo_store/shop/templates/shop/cart.html

satchmo_store/shop/templates/admin/base_site.html

satchmo_store/shop/templates/admin/index.html

satchmo_store/contact/templates/comments/form.html

satchmo_store/contact/templates/comments/preview.html

satchmo_store/contact/templates/comments/posted.html

satchmo_store/contact/templates/admin/contact/order/change_form.html

satchmo_store/contact/templates/contact/view_profile.html

satchmo_store/contact/templates/contact/update_form.html

satchmo_store/contact/templates/contact/login_signup_address.html

satchmo_store/contact/templates/contact/_country_match_script.html

satchmo_store/contact/templates/contact/login_signup.html

satchmo_store/contact/templates/contact/_addressblock.html

satchmo_store/contact/templates/registration/registration_form.html

satchmo_store/contact/templates/registration/account_info.html

satchmo_store/contact/templates/registration/registration_complete.html

satchmo_store/contact/templates/registration/password_reset_done.html

satchmo_store/contact/templates/registration/activate.html

satchmo_store/contact/templates/registration/repeat_activation.html

satchmo_store/contact/templates/registration/login.html

satchmo_store/contact/templates/registration/password_change_form.html

satchmo_store/contact/templates/registration/password_reset_form.html

satchmo_store/contact/templates/registration/logout.html

satchmo_store/contact/templates/registration/password_change_done.html

Livesettings

livesettings/templates/livesettings/site_settings.html

livesettings/templates/livesettings/group_settings.html

livesettings/templates/livesettings/_admin_site_views.html

Shipping

shipping/templates/shipping/options.html

Satchmo_utils

satchmo_utils/thumbnail/templates/widget/file.html

4.8 Satchmo Template Filters and Tags

Filters and Tags provide great flexibility when working with templates and are an important part of Django, and by association, Satchmo.

It is possible to easily create your own filters and tags (you can read the Django Template documentation for details), but Satchmo also includes several useful filters and tags already.

4.8.1 Filter Reference

To use Tags in templates, you need to use the `load` tag to load the custom library into the template.

discount_cart_total

Returns the discounted total for a given cart item.

> Usage:

```
{% load satchmo_discounts %}
{{ cart|discount_cart_total:"discount" }}
```

The `discount` argument, if omitted will only result in returning the total for the cart, not the discounted total.

discount_line_total

Returns the discounted line total for this cart item.

> Usage:

```
{% load satchmo_discounts %}
{{ product|discount_line_total:"sale" }}
```

discount_price

Returns the product price with the discount applied.

> Usage:

```
{% load satchmo_discounts %}
{{ product|discount_price:"sale" }}
```

You would replace `product` with your actual product template variable.

discount_ratio

Returns the discount as a ratio, making sure that the percent is under 1.

> Usage:

```
{% load satchmo_discounts %}
{{ discount|discount_ratio }}
```

discount_saved

Returns the amount saved by the discount.

> Usage:

```
{% load satchmo_discounts %}
{{ product|discount_saved:"sale" }}
```

sale_price

Returns the product unit price with the best auto discount applied.

> Usage:

```
{% load satchmo_discounts %}
{{ product|sale_price }}
```

This filter finds all applicable *Site-wide Sales via Auto Discounts*, then sorts them by percentage. The return value would represent the greatest discount available for a product.

4.8.2 Tag Reference

To use Tags in templates, you need to use the `load` tag to load the custom library into the template.

filter_admin_app_list

Filters the list of installed applications returned by `django.contrib.admin.templatetags.adminapplist`, excluding applications installed by Satchmo.

> Usage:

```
{% load satchmo_adminapplist %}
{% filter_admin_app_list app_list varname %}
```

In the above usage you would replace `app_list` with the list of applications you would like to filter, and `varname` with the name of the variable you want to be returned in the template context.

inprocess_order_list

Returns a formatted list of in-process orders.

> Usage:

```
{% load satchmo_adminorder_tags %}
{% inprocess_order_list %}
```

This tag renders another template (default is `templates\admin\_ordercount_list.html`), which produces an html list, using the `django.template.Library().inclusion_tag` library.

product_upsell

Allows for easy upselling or cross-selling of a product. Up-selling can imply selling something additional, or selling something that is more profitable or otherwise preferable for the seller instead of the original sale [1] .

A practical example would be If you were selling ebooks, you could make a checkbox on the ebook detail page, which would allow your customer to order the companion CD. This is useful as you are able to provide customers with a direct way of purchasing a companion item without the need to search for it.

> Usage:

```
{% load satchmo_upsell %}
{% product_upsell product %}
```

This tag renders another template (default is `templates\upsell\product_upsell.html`) using the `django.template.Library().inclusion_tag` library. The tag should be used within a html form, most commonly within the order form for the product you want to upsell.

No product options

Currently, the goal product doesn't have any options. It can be a be a product variation, but can't be a configurable product by itself.

4.9 Satchmo Views

4.9.1 URLs by Name

Here's an overview showing the name and view for URLs in Satchmo.

URL paths are relative to SHOP_BASE.

satchmo_shop_home

> **URL** /
>
> **View** `satchmo_store.shop.views.home.home()`

satchmo_category_index

> **URL** /<*CATEGORY_SLUG*>/

---

[1] Wikipedia article Up-selling

> View `product.views.category_index()`

satchmo_category

> **URL** /<*CATEGORY_SLUG*>/slug1/slug2/.../<slug>/

> **View** `product.views.category_view()`

satchmo_product

> **URL** /<*PRODUCT_SLUG*>/<slug>/

> **View** `product.views.get_product()`

satchmo_cart

> **URL** /cart/

> **View** `satchmo_store.shop.views.cart.display()`

satchmo_checkout-step1

> **URL** /checkout/

> **View** `payment.views.contact.contact_info_view()`

satchmo_balance_remaining

> **URL** /checkout/balance/

> **View** `payment.views.balance.balance_remaining_order()`

satchmo_balance_remaining_order

> **URL** /checkout/balance/<id>/

> **View** `payment.views.balance.balance_remaining()`

satchmo_charge_remaining

> **URL** /checkout/custom/charge/

> **View** `payment.views.balance.charge_remaining_post()`

satchmo_charge_remaining_post

> **URL** /checkout/custom/charge/<id>/

> **View** `payment.views.balance.charge_remaining()`

satchmo_checkout_auth_required

> **URL** /checkout/mustlogin/

> **View** `payment.views.contact.authentication_required()`

satchmo_checkout-success

> **URL** /checkout/success/

> **View** `payment.views.checkout.success()`

satchmo_contact

> **URL** /contact/

> **View** `satchmo_store.shop.views.contact.form()`

satchmo_contact_thanks

> **URL** /contact/thankyou/

> **View** *Contact thanks*

satchmo_order_history

> **URL** /history/
>
> **View** `satchmo_store.shop.views.orders.order_history()`

satchmo_quick_order

> **URL** /quickorder/
>
> **View** `satchmo_store.shop.views.cart.add_multiple()`

satchmo_search

> **URL** /search/
>
> **View** `satchmo_store.shop.views.search.search_view()`

satchmo_order_tracking

> **URL** /tracking/
>
> **View** `satchmo_store.shop.views.orders.order_tracking()`

4.9.2 Use of Generic Django Views

Contact thanks

The view `satchmo_store.shop.views.contact.form()` redirects the user to this page after sending the email successfully.

> **url** `satchmo_contact_thanks`
>
> **template** shop/contact_thanks.html
>
> **context** None

4.9.3 Views by App

payment

`payment.views.balance.`**`balance_remaining`**`()`

> **Url** `satchmo_balance_remaining`
>
> **Template** shop/checkout/balance_remaining.html
>
> **Context**
>
> > **form** an instance of `payment.forms.PaymentMethodForm`
> >
> > **paymentmethod_ct** the length of the return value of payment.active_gateways()
> >
> > **order**

`payment.views.balance.`**`balance_remaining_order`**`()`

> **Url** `satchmo_balance_remaining_order`
>
> **Template** shop/checkout/balance_remaining.html

> > **Context** see *balance_remaining's context*.

`payment.views.balance.`**`charge_remaining`**`()`
> Displays a confirmation form for the order with id *id*.

> > **Url** `satchmo_charge_remaining_post`

> > **Template** payment/admin/charge_remaining_confirm.html

> > **Context**

> > > **form** an instance of `payment.forms.CustomChargeForm`

`payment.views.balance.`**`charge_remaining_post`**`()`
> Handles the submit response to `payment.views.balance.charge_remaining()`.

> > **Url** `satchmo_charge_remaining`

> > **Template** payment/admin/charge_remaining_confirm.html

> > **Context**

> > > **form** an instance of `payment.forms.CustomChargeForm`

`payment.views.contact.`**`authentication_required`**`()`

> > **Url** `satchmo_checkout_auth_required`

> > **Template** shop/checkout/authentication_required.html

> > **Context** None

`payment.views.contact.`**`contact_info_view`**`()`

> > **Url** *satchmo_checkout-step1*

> > **Template** shop/checkout/form.html

> > **Context**

> > > **form** an instance of `payment.forms.PaymentContactInfoForm`

> > > **country**

> > > **paymentmethod_ct** the length of the return value of payment.active_gateways(), passed through the payment_choices signal

product

`product.views.`**`get_product`**`()`

> > **Url** `satchmo_product`

> > **Template** product/product.html

> > **Context**

> > > **product** the product with slug *slug*

> > > **current_product** clone of *product* for product variations

> > > **default_view_tax** see `default_view_tax`.

> > > **sale** see `sale`.

> > > **error_message** errors

satchmo_store

satchmo_store.shop.views.cart.**add_multiple**()

>   **Url** satchmo_quick_order

>   **Template** shop/multiple_product_form.html

>   **Context**

>   >   **form** an instance of satchmo_store.shop.forms.MultipleProductForm

satchmo_store.shop.views.cart.**display**()

>   **Url** satchmo_cart

>   **Template** shop/cart.html

>   **Context**

>   >   **cart** cart

>   >   **default_view_tax** see default_view_tax.

>   >   **error_message** errors

>   >   **sale** see sale.

satchmo_store.shop.views.contact.**form**()

>   **Url** satchmo_contact

>   **Template** shop/contact_form.html

>   **Context**

>   >   **form** an instance of satchmo_store.shop.views.contact.ContactForm

satchmo_store.shop.views.home.**home**()

>   **Url** satchmo_shop_home

>   **Template** shop/index.html

>   **Context**

>   >   **all_products_list**

>   >   **is_paginated**

>   >   **page_obj**

>   >   **paginator**

satchmo_store.shop.views.orders.**order_history**()

>   **Url** satchmo_order_history

>   **Template** shop/order_history.html

>   **Context**

>   >   **contact**

>   >   **default_view_tax** see default_view_tax.

>   >   **orders**

satchmo_store.shop.views.orders.**order_tracking**()

>   **Url** satchmo_order_tracking

> **Template** shop/order_tracking.html
>
> **Context**
>
> > **contact**
> >
> > **default_view_tax** see `default_view_tax`.
> >
> > **order**

satchmo_store.shop.views.search.**search_view**()

> **Url** `satchmo_search`
>
> **Template** shop/search.html
>
> **Context**

4.9.4 Payment Processor Views

PayPal

URLs by Name

URLs are relative to /checkout/`PAYMENT_PAYPAL.URL_BASE`/.

PAYPAL_satchmo_checkout-step2

> **URL** /
>
> **View** `payment.modules.paypal.views.pay_ship_info()`

PAYPAL_satchmo_checkout-step3

> **URL** /confirm/
>
> **View** `payment.modules.paypal.views.confirm_info()`

PAYPAL_satchmo_checkout_free-confirm

> **URL** /confirmorder/
>
> **View** payment.views.confirm.confirm_free_order

PAYPAL_satchmo_checkout-ipn

> **URL** /ipn/
>
> **View** payment.modules.paypal.views.ipn

PAYPAL_satchmo_checkout-success

> **URL** /success/
>
> **View** `payment.views.checkout.success()`

Views

payment.modules.paypal.views.**pay_ship_info**()

> **Url** *PAYPAL_satchmo_checkout-step2*
>
> **Template** shop/checkout/paypal/pay_ship.html
>
> **Context**

> > **form** an instance of payment.views.payship.simple_pay_ship_process_form
>
> > **PAYMENT_LIVE** whether the Paypal module is 'live'

payment.modules.paypal.views.**confirm_info**()

> **Url** *PAYPAL_satchmo_checkout-step3*
>
> **Template** shop/checkout/paypal/confirm.html
>
> **Context**
>
> > **order** the instance of satchmo_store.shop.models.Order being paid for.
> >
> > **post_url** either `PAYMENT_PAYPAL.POST_URL` or `PAYMENT_PAYPAL.POST_TEST_URL`, depending on whether the Paypal module is `'live'`.
> >
> > **default_view_tax** see `default_view_tax`.
> >
> > **business** either `PAYMENT_PAYPAL.BUSINESS` or `PAYMENT_PAYPAL.BUSINESS_TEST`, depending on whether the Paypal module is `'live'`.
> >
> > **currency_code** the value of `PAYMENT_PAYPAL.CURRENCY_CODE`.
> >
> > **return_address** the value of `PAYMENT_PAYPAL.RETURN_ADDRESS`.
> >
> > **invoice** the id of *order*
> >
> > **subscription**
> >
> > **PAYMENT_LIVE** whether the Paypal module is `'live'`.

4.9.5 Forms

class satchmo_store.shop.views.contact.**ContactForm**

> **contents**
>   a CharField with a TextArea widget.
>
> **inquiry**
>
> > **a ChoiceField with choices:**
> >
> > - General Question
> > - Order Problem
>
> **name**
>   a CharField.
>
> **sender**
>   an EmailField
>
> **subject**
>   a CharField.

class payment.forms.**CustomChargeForm**

> **amount**
>   a DecimalField.

> **notes**
>> a CharField.
>
> **orderitem**
>> a hidden IntegerField field.
>
> **shipping**
>> a DecimalField.

class satchmo_store.shop.forms.**MultipleProductForm**
> A form populated with PositiveRoundedDecimalField fields, one for each active product.

class payment.forms.**PaymentContactInfoForm**
> Subclasses payment.forms.PaymentMethodForm
>
> and satchmo_store.contact.forms.ContactInfoForm.

class payment.forms.**PaymentMethodForm**

> **paymentmethod**
>> a radio-button selection with choices payment.config.labelled_gateway_choices().

4.9.6 Context variables

Variables provided by Satchmo's context processor

Here are the variables provided in the context by Satchmo's context processor, satchmo_store.shop.context_processor.settings.

> **shop_base** The base URL from SHOP_BASE in SATCHMO_SETTINGS
>
> **shop** an object satchmo_store.shop.models.Config representing the current store configuration
>
> **shop_name** the *store_name* value from the config
>
> **media_url** The current media url, taking into account SSL
>
> **STATIC_URL** The current static url, taking into account SSL
>
> **cart_count** number of items in the cart
>
> **cart** user's current cart; an instance of satchmo_store.shop.models.Cart
>
> **categories** all categories in the site
>
> **is_secure** A boolean representing weather or not SSL is enabled
>
> **request** The HttpRequest object passed into the context processor
>
> **login_url** the LOGIN_URL setting in your settings.py
>
> **logout_url** the LOGOUT_URL setting in your settings.py
>
> **storewide_sale** An instance of product.models.Discount if there is a global sale, or None

Common variables

default_view_tax
> The configuration value with group 'TAX' and key 'DEFAULT_VIEW_TAX' from livesettings.

sale
> The return value of product.utils.find_best_auto_discount(), which is the discount with the highest discount percentage.

4.10 Signals in Satchmo

Signals are a very powerful tool available in Django that allows you to decouple aspects of your application. The Django Signals Documentation, has this summary:

> "In a nutshell, signals allow certain senders to notify a set of receivers that some action has taken place."

In addition to all of the built in Django signals, Satchmo includes a number of store related signals. By using these signals, you can add very unique customizations to your store without needing to modify the Satchmo code.

4.10.1 Signal Descriptions

satchmo_store.shop.signals.**order_success**(*sender*, *order=None*, ***kwargs*)
> Base class for all signals

> Internal attributes:

>> **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.shop.signals.**order_cancel_query**(*sender*, *order=None*, ***kwargs*)
> Base class for all signals

> Internal attributes:

>> **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.shop.signals.**order_cancelled**(*sender*, *order=None*, ***kwargs*)
> Base class for all signals

> Internal attributes:

>> **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.shop.signals.**satchmo_cart_add_complete**(*sender*, *cart=None*, *cartitem=None*, *product=None*, *request=None*, *form=None*, ***kwargs*)
> Base class for all signals

> Internal attributes:

>> **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.shop.signals.**satchmo_cart_add_verify**(*sender*, *cart=None*, *cartitem=None*, *added_quantity=None*, *details=None*, ***kwargs*)
> Base class for all signals

> Internal attributes:

>> **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.shop.signals.**satchmo_cart_changed**(*sender*, *cart=None*, *request=None*, ***kwargs*)
> Base class for all signals

Internal attributes:

>   **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.shop.signals.**satchmo_cartitem_price_query**(*sender*, *cartitem=None*, ***kwargs*)

>   Base class for all signals

Internal attributes:

>   **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.shop.signals.**satchmo_cart_details_query**(*sender*, *product=None*, *quantity=None*, *details=None*, *request=None*, *form=None*, ***kwargs*)

>   Base class for all signals

Internal attributes:

>   **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.shop.signals.**satchmo_post_copy_item_to_order**(*sender*, *cartitem=None*, *order=None*, *orderitem=None*, ***kwargs*)

>   Base class for all signals

Internal attributes:

>   **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.shop.signals.**satchmo_context**(*sender*, *context=None*, ***kwargs*)

>   Base class for all signals

Internal attributes:

>   **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.shop.signals.**cart_add_view**(*sender*, *request=None*, *method=None*, ***kwargs*)

>   Base class for all signals

Internal attributes:

>   **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.shop.signals.**sendfile_url_for_file**(*sender*, *file=None*, *product=None*, *url_dict={}*, ***kwargs*)

>   Base class for all signals

Internal attributes:

>   **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.shop.signals.**rendering_store_mail**(*sender*, *send_mail_args={}*, *context={}*, ***kwargs*)

>   Base class for all signals

Internal attributes:

>   **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.shop.signals.**sending_store_mail**(*sender*, *send_mail_args={}*, *context={}*, ***kwargs*)

>   Base class for all signals

Internal attributes:

> **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.contact.signals.**satchmo_contact_view**(*sender*, *contact=None*, *contact_dict=None*, ***kwargs*)

Base class for all signals

Internal attributes:

> **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.contact.signals.**satchmo_contact_location_changed**(*sender*, *contact=None*, ***kwargs*)

Base class for all signals

Internal attributes:

> **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.contact.signals.**validate_postcode**(*sender*, *postcode=None*, *country=None*, ***kwargs*)

Base class for all signals

Internal attributes:

> **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.accounts.signals.**satchmo_registration**(*sender*, *contact=None*, *subscribed=None*, *data=None*, ***kwargs*)

Base class for all signals

Internal attributes:

> **receivers** { receriverkey (id) : weakref(receiver) }

satchmo_store.accounts.signals.**satchmo_registration_verified**(*sender*, *contact=None*, ***kwargs*)

Base class for all signals

Internal attributes:

> **receivers** { receriverkey (id) : weakref(receiver) }

payment.signals.**confirm_sanity_check**(*sender*, *controller=None*, ***kwargs*)

Base class for all signals

Internal attributes:

> **receivers** { receriverkey (id) : weakref(receiver) }

payment.signals.**payment_methods_query**(*sender*, *methods=None*, *cart=None*, *order=None*, *contact=None*, ***kwargs*)

Base class for all signals

Internal attributes:

> **receivers** { receriverkey (id) : weakref(receiver) }

payment.signals.**payment_choices**(*sender*, *choices=None*, ***kwargs*)

Base class for all signals

Internal attributes:

> **receivers** { receriverkey (id) : weakref(receiver) }

product.signals.**index_prerender**(*sender*, *request=None*, *context=None*, *category=None*, *brand=None*, *object_list=None*, ***kwargs*)

> Base class for all signals
>
> Internal attributes:
>
> > **receivers** { receriverkey (id) : weakref(receiver) }

product.signals.**satchmo_price_query**(*sender*, *price=None*, *slug=None*, *discountable=None*, ***kwargs*)

> Base class for all signals
>
> Internal attributes:
>
> > **receivers** { receriverkey (id) : weakref(receiver) }

product.signals.**subtype_order_success**(*sender*, *product=None*, *order=None*, *subtype=None*, ***kwargs*)

> Base class for all signals
>
> Internal attributes:
>
> > **receivers** { receriverkey (id) : weakref(receiver) }

product.signals.**discount_filter_items**(*sender*, *discounted=None*, *order=None*, ***kwargs*)
> Base class for all signals
>
> Internal attributes:
>
> > **receivers** { receriverkey (id) : weakref(receiver) }

4.10.2 External Signals Used in Satchmo

Satchmo depends on signals in `satchmo_utils.signals` and triggers them at various points of execution; below are some of them.

satchmo_utils.signals.**application_search**()

Sent by [satchmo_store.shop.views.search.search_view()](#) to ask all listeners to add search results.

Arguments sent with this signal:

> **sender** The `product.models.Product` model (Note: not an instance of Product)
>
> **request** The `HttpRequest` object used in the search view
>
> **category** The category slug to limit a search to a specific category
>
> **keywords** A list of keywords search for
>
> **results** A dictionary of results to update with search results. The contents of the dictionary should contain the following information:
>
> > **categories** A `QuerySet` of `product.models.Cateogry` objects which matched the search criteria
> >
> > **products** A `Queryset` of `product.models.Product` objects which matched the search critera

satchmo_utils.signals.**collect_urls**()

Sent by urls modules to allow listeners to add or replace urls to that module

Arguments sent with this signal:

sender The module having url patterns added to it

patterns The url patterns to be added. This is an instance of `django.conf.urls.patterns`

section The name of the section adding the urls (Note: this argument is not always provided). For example '__init__' or 'product'

Example:

```python
from satchmo_store.shop.signals import satchmo_cart_add_complete
import myviews

satchmo_cart_add_complete.connect(myviews.cart_add_listener, sender=None)
```

satchmo_utils.signals.**form_init**()

Sent when a contact info form is initialized. Contact info forms include:

- `contact.forms.ContactInfoForm`
- `contact.forms.ExtendedContactInfoForm`
- `payment.forms.PaymentContactInfoForm`

Arguments sent with this signal:

sender The model of the form being initialized. The value of sender will be one of the models defined above.

form An instance of the form (whose type is defined by `sender`) being intitialized.

See Ensuring Acceptance of Terms during Checkout for an example of how this signal can be used.

satchmo_utils.signals.**form_postsave**()

Sent after a form has been saved to the database

Arguments sent with this signal:

sender The form model of the form being set (Note: Not an instance). Possible values include:

- `satchmo_store.contact.forms.ContactInfoForm`
- `payment.modules.purchaseorder.forms.PurchaseorderPayShipForm`
- `payment.forms.CreditPayShipForm`
- `payment.forms.SimplePayShipForm`
- `payment.forms.PaymentContactInfoForm`

form

- The instance of the form defined by one of the above models that was saved.

object

- A `satchmo_store.contact.models.Contact` instance if the form being saved is an instance of `satchmo_store.contact.forms.ContactInfoForm` otherwise this value does not exist.

formdata

- The data associated with the form if the form being saved is an instance of `satchmo_store.contact.forms.ContactInfoForm` otherwise this value does not exist.

4.10.3 Examples

Putting it All Together

This section contains a brief example of how to use signals in your application. For this example, we want to have certain products that are only available to members. Everyone can see the products, but only members can add to the cart. If a non-member tries to purchase a product, they will get a clear error message letting them know they need to be a member.

The first thing to do is create a *listeners.py* file in your app. In this case, the file would look something like this:

```python
"""
A custom listener that will evaluate whether or not the product being added
to the cart is available to the current user based on their membership.
"""
from satchmo_store.shop.exceptions import CartAddProhibited
from django.utils.translation import gettext_lazy as _

class ContactCannotOrder(CartAddProhibited):
    def __init__(self, contact, product, msg):
        super(ContactCannotOrder, self).__init__(product, msg)
        self.contact = contact

def veto_for_non_members(sender, cartitem=None, added_quantity=0, **kwargs):
    from utils import can_user_buy
    customer = kwargs['cart'].customer
    if can_user_buy(cartitem.product, customer):
        return True
    else:
        msg = _("Only members are allowed to purchase this product.")
        raise ContactCannotOrder(customer, cartitem.product, msg)
```

Next, you need to create the *can_user_buy* function. Your *utils.py* file could look something like this (details left up to the reader):

```python
def can_user_buy(product, contact=None):
    """
    Given a product and a user, return True if that person can buy it and
    False if they can not.
    This doesn't work as it stands now. You'll need to customize the
    is_member function
    """
    if is_member(contact):
        return True
    else:
        return False
```

The final step is to make sure your new listener is hooked up. In your *models.py* add the following code:

```python
from listeners import veto_for_non_members
from satchmo_store.shop import signals

signals.satchmo_cart_add_verify.connect(veto_for_non_members, sender=None)
```

Now, you should be able to restrict certain products to only your members. The nice thing is that you've done this without modifying your satchmo base code.

Ensuring Acceptance of Terms during Checkout

The signal `satchmo_utils.signals.form_init()` can be combined with the `payment.listeners.form_terms_listener` to add a custom terms and conditions acceptance box into your checkout flow.

First, add the terms view in your `/localsite/urls.py` file:

```
urlpatterns += patterns('',
url(r'^shop_terms/$', 'project-name.localsite.views.shop_terms',
    name="shop_terms"),
)
```

Next, create the view in your `/localsite/views.py` to display the terms:

```python
from django.shortcuts import render_to_response
from django.template import RequestContext


def shop_terms(request):
    ctx = RequestContext(request, {})
    return render_to_response('localsite/shop-terms.html',
        context_instance=ctx)
```

Now, you will need modify the checkout html to display the new form. Copy `/satchmo/apps/payment/templates/shop/checkout/pay_ship.html` to `/project-name/templates/shop/checkout/pay_ship.html`.

Add the following code to the copied `pay_ship.html` to display the form:

```
{{ form.terms }} {{ form.terms.label|safe }}
{% if form.terms.errors %}<br/>**{{ form.terms.errors|join:", " }}{% endif %}
```

Make sure you register the forms_terms_listener by adding the following code to your `/localsite/models.py`:

```python
from payment.forms import SimplePayShipForm
from payment.listeners import form_terms_listener
from satchmo_utils.signals import form_init


form_init.connect(form_terms_listener, sender=SimplePayShipForm)
```

The final step is to create your actual `store-name/templates/localsite/shop-terms.html`, like this:

```
{% extends "base.html" %}
{% block content %}
<p>Put all of your sample terms here.</p>
{% endblock %}
```

Now, when users checkout, they must agree to your store's terms.

Optional Modules

5.1 Google Base

Google Base is a "free Google service that lets you publish virtually any kind of information." This module allows you to create a feed of your products and add to Google Base. It is not enabled by default.

5.1.1 Features

- It exports all active products into the feed.

- It adds all the legal payment types to the feed.

- It skips "Configurable Products" - since the products that should be exported are child ProductVariations. (I.E. the XL Blue Fancy Shirt, not the Fancy Shirt parent)

- It adds dimensions if found.

- It automatically tries to figure out how to apply product options. For example, if you have an OptionGroup named "Size" or "Sizes", it will automatically be applied to the GoogleBase attribute <g:size>size</g:size>. Similarly for "Color", "Style", etc.

- It applies the same logic to attributes. If you have an attribute, "Brand", for example, then the feed will contain an entry for "<g:brand>brand</g:brand>"

- In either case, if it is not found in the base google tags, it will make a custom tag. For example. "<c:book_type:string>hardback</c:book_type:string>"

5.1.2 Installing

On your site:

- Add "satchmo_ext.product_feeds" to your settings.py INSTALLED_APPS

On Google:

> Review the Google Installation Steps

5.2 Upsell

Allows for easy upselling or cross-selling of a product. Up-selling can imply selling something additional, or selling something that is more profitable or otherwise preferable for the seller instead of the original sale [1] .

A practical example would be If you were selling ebooks, you could make a checkbox on the ebook detail page, which would allow your customer to order the companion CD. This is useful as you are able to provide customers with a direct way of purchasing a companion item without the need to search for it.

Note: Currently, the goal product doesn't have any options. It can be a be a product variation, but can't be a configurable product by itself.

When entering a category in the admin interface, you will have several fields to fill in. Below is a description of each field and how it is used:

Target Product The product(s) for which you want to add an upsell to.

Goal Product This is the product you are upselling.

Upsell Style

> **There are five styles:**
>
> > • Checkbox to add 1.
> >
> > • Checkbox to add 1 (checked by default).
> >
> > • Checkbox to match the quantity of the product on the page.
> >
> > • Checkbox to match the quantity of the product on the page (checked selected by default).
> >
> > • Simple manual entry quantity box.

Notes Optional field for internal notes.

Creation Date Date when upsell was created.

Translations Optional field for translated description(s).

5.2.1 A short tutorial

Here is a quick step by step overview for adding and using the upsell application.

• In your site's `settings.py` file, edit the `INSTALLED_APP` list to include `satchmo_ext.upsell`. Run a syncdb on your site to ensure that the necessary database tables are created.

• Log into the your site's admin interface, and add a new product. You will want to add the target product (remember that an upselling item can be a standalone product, as well as a variation of an existing product) and an upsell product. You can read the *Satchmo products* page for details on how to do this.

• From your site's admin main page, navigate to the Upsells section to add/change an upsell.

5.2.2 Useful links

• *Upsell template tag*

• *Satchmo products*

---

[1] Wikipedia article on Up-selling

5.2.3 References

Tutorials

6.1 Adding a Product to Your Store

This tutorial will walk through the typical process a store administrator will use to create a new product.

For this example, we will be adding the following product to our store:

Full Name: Adventurer Hat

Description: This well made, extremely durable hat can be used in all sorts of conditions. Whether you are a weekend adventurer or career explorer, you will be well served by this product. It can keep you warm in the high mountains and keep the sun at bay in the middle of the desert. Buy one today and start exploring!

Sizes: Small, Medium, Large

Colors: Black, Brown, White

Price: $49.95

6.1.1 Steps

Setup product types

Verify that the necessary module `product.modules.configurable` is enabled in `INSTALLED_APPS` which is satisfied by default.

Create the Product

From the main admin screen, under the Product section, click the "Add" icon next to "Products".

Product			Orders in Process	
Categories	✚ Add	✎ Change	View all Orders	
Configurable Products	✚ Add	✎ Change	Admin Tools (hide)	
Custom Products	✚ Add	✎ Change	▪ Edit Site Settings	
Custom text fields	✚ Add	✎ Change	▪ Edit Inventory	
Discounts	✚ Add	✎ Change	▪ Export Product Defs	
Downloadable Products	✚ Add	✎ Change	▪ Product Variation Manager	
Option Groups	✚ Add	✎ Change	▪ Cache Status	
Option Items	✚ Add	✎ Change		
Product variations	✚ Add	✎ Change		
Products	✚ Add	✎ Change		
Subscription Products	✚ Add	✎ Change		
Tax Classes	✚ Add	✎ Change		

The main product screen will now be shown. There are multiple fields available to customize the product. We will fill in some of the most important.

If you do not already have a "hats" category in place, create one by clicking on the green plus icon above categories. Once it is selected, you should see a screen like this:

Home › Product › Products › Add Product

Add Product

Site:	localhost ▾ ✚

Category: ✚

Available Category

🔍 [   ]

Books
Shirts
Software
Shirts :: Short Sleeve
Books :: Fiction
Books :: Non Fiction
Books :: Fiction :: Science Fiction

Chosen Category

Select your choice(s) and click ➡

hats

➡
➡

● Choose all

○ Clear all

Now, create the description for the product. The sku and slug fields can be left blank and will be automatically populated with values:

Satchmo allows you to track inventory of your products, feature certain products and set whether or not the products are shippable. You may configure these items in the next section of the Product page.

Additionally, you have the option to configure the Meta options (used for Search Engine Optimization), dimensions and tax properties. Depending on your product and store location you may or may not need to set these items. For this example, we will use the default values.

Date added: | | Today | 📅

☑ Is product active?

This will determine whether or not this product will appear on the site

☐ Featured Item

Featured items will show on the front page

Number in stock: | 0 |

Total sold: | 0 |

Ordering: | 0 |

Override alphabetical order in category display

Shipping: | Default ▼ |

If this is 'Default', then we'll use the product type to determine if it is shippable.

Meta Data (Show)

Item Dimensions (Show)

Tax (Show)

Related Products (Show)

All products will need a price and an image. You can configure volume discounts if you like, but for this example, we will set the hat's price to $49.95. We will also upload the image.

Product Attributes

Language	Attribute Name	Value	Delete?
-------- ▼			

Prices

Price	Discount Quantity	Expires	Delete?
$ 49.95	1.0		Today | 📅
$	1.0		Today | 📅

Product Images

Product Image: #1

Picture: /home/chris/Pictures/adven [Browse...]

Optional caption:

Sort Order: 0

Once all off the product's information has been entered, you may save it.

At this point, you have created the Adventurer's hat but have not created the various size and color combinations you would like to offer. Instead of creating 9 new products (one for each combo), we can use Product Variations to make our lives a little bit easier.

Create the Option Groups

Going back to the main admin screen, click the "Add" Icon next to "Option Groups."

In the option group screen, we will set up our "Hat Size" option group. In this example, we configure the sizes so that a Large hat will be $2.00 extra.

After the values, are entered, click save.

We will repeat the same process to create the hat colors options. Once they are both entered, you should see them in the admin interface.

Create The Product Variations

Now that we have our hat created and have the colors and sizes in place, we need to create hats with each color and size. To do this, we go back to our Product and configure it to use these options.

From the main admin screen, click on "Products." You should have a list of all your store's products:

After displaying the Adventurer's Hat, scroll to the bottom of the page where you can configure the subtypes associated with the hat. Click on "Add Configurable Product":

On the configurable product screen, highlight the "Hat Size" and "Hat Color" options and click "Save."

Select the "Adventurer Hat" configurable product:

From the screen, click on the "Create Variations" box and the click "Save" to associate the new variations with the Product.

Change Configurable Product

| Product: | Adventurer Hat |

☑ Create Variations

Create ProductVariations for all this product's options. To use this, you must fir

Option Group:

```
Colors - Hat Colors
Size - Hat Size
Book type
sizes
colors
Full/Upgrade
```
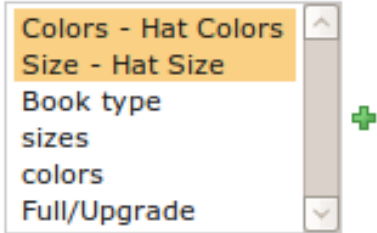
Hold down "Control", or "Command" on a Mac, to select more than one.

Helpers

- Variation Manager

Variations

Black, Large	➕ Add Variation
Black, Medium	➕ Add Variation
Black, Small	➕ Add Variation
Brown, Large	➕ Add Variation
Brown, Medium	➕ Add Variation
Brown, Small	➕ Add Variation
White, Large	➕ Add Variation

You have now created all 9 variations of this product!

View the Final Product

You can view your final product here - /product/adventure-hat/

In this example, note how the price automatically changes based on the options selected.

Adventurer Hat

This well made, extremely durable hat can be used in all sorts of conditions. Whether you are a weekend adventurer or career explorer, you will be well served by this product. It can keep you warm in the high mountains and keep the sun at bay in the middle of the desert. Buy one today and start exploring!

Price:

$51.95

Please choose your options:

Size | Large (+ $2.00) ⌄ | Colors | Brown ⌄ | Quantity | 1 | Add to cart

Deploying A Store

7.1 Deploying A Store

7.1.1 PCI-DSS Compliance

Any website that collects credit card payment information is subject to the Payment Card Industry Data Security Standard (PCI-DSS). PCI compliance is a complicated topic; you need to understand it fully and know how to use Satchmo in a PCI-DSS compliant manner.

If you are deploying a store with Satchmo, it is your responsibility to make sure you understand all the rules and how to apply them to your situation. The Wikipedia article on PCI provides a good overview and links to additional information.

According to the PCI-DSS: "PCI DSS requirements are applicable if a Primary Account Number (PAN) is stored, processed, or transmitted. If a PAN is not stored, processed, or transmitted, PCI DSS requirements do not apply."

Any payment module that collects credit card information will put your server under PCI scope, and require you to comply with PCI-DSS requirements. The exception to this is a module like PayPal or Google Checkout, where the credit card information is actually entered on PayPal's site. If the credit card information is posted to your server, you are under PCI scope. (It doesn't matter whether you store the information or not. Simply transmitting it is enough.)

> **Warning:** It is your responsibility to make sure Satchmo is deployed in a PCI-DSS compliant manner.

7.1.2 Additional Notes

Once your store has been successfully created in your testing environment, you will want to keep a few things in mind when deploying into production:

- Follow the notes in the Django deployment guide for your server setup.

- Double check and make sure that you have DEBUG = False in your settings

- Disable your test urls for your payment modules.

> **Note:** Remember, after saving changes to your payment processor, you will need to restart your server for the changes to take effect.

- Run a small transaction through the live store to verify payment works.

- Let the world know about your store by submitting it here

7.2 Managing Your Store

Once you have your store up and running, Satchmo has several tools that will make it easier for you to administer and maintain your site.

7.2.1 Admin Toolbar

Satchmo includes a very useful Administrative toolbar that allows the store administrator to browse the active store and displays useful statistics about your store including:

- Current Satchmo version

- Link to the admin site

- Number of and link to new orders

- Number of carts in the past, hour, day and week

- Number of contacts in the past week as well as total

- If viewing a specific item, it will also show the number of items sold

In order for this to work you must have `satchmo_ext.satchmo_toolbar` in your installed apps.

7.2.2 Editing Items Bookmarklet

As described in this article , Django supports a useful bookmarklet that allows you to quickly jump to the admin page for a specific item in the store. In order to use this feature, make sure that you drag the necessary bookmarklet from `/admin/doc/bookmarklets` to your browser's bookmark bar. Then whenever you are on a product page which can be editable from the admin, you can quickly jump there.

7.2.3 Users and Groups

In its default setting, Satchmo does not come with any predefined groups. What this means for you is that any user that has staff permissions will be able to see and edit all of the items in the admin. For production use, you will probably want to create several custom groups so that users can have different access based on their needs. The Django documentation can help you with the process of setting up individual user as well as group permissions.

Here is one scenario that you could use for configuring your groups:

- Store Admin (full access to the admin site)

- Order Manager (able to see and manage orders)

- Product Manager (add, delete and edit product information)

- Contact Manager (add, delete and edit contact information)

Additional Notes

8.1 Satchmo Development Branch

These are the notes for the development version of Satchmo.

8.1.1 New Features

- upgraded to use the latest Sorl Thumbnail code. Please review the migration notes
- Added Paypal Payflowpro payment module
- Updated all templates to use Django static files
- Send email to gift certificate recipients
- Rewrite PDF (document) generation, removed trml2pdf and Reportlab dependencies: please see *trml2pdf* and the migration notes when upgrading
- Updated to using django-registration 0.8

8.1.2 Translations

- Italian updates
- Russian updates

8.1.3 Bug Fixes

- Make variation manager more robust so that you can't delete an order when deleting the variation.
- Javascript updates for newer jquery
- Fix bug with required login not working
- Ensure TRACK_INVENTORY is used in all payment modules
- Updated cache settings to use the new format
- Multiple minor bug cleanups and fixes
- Update views to use new class-based generic views
- Update Satchmo to use Django's timezone support

- Fixes to thumbnail widget paths
- Fix shipping tax config

8.1.4 Performance Improvements

- DB optimizations using new prefetch_related from Django 1.4
- Tweaks to product ratings

8.1.5 Additional Notes

- Remove django-signals-ahoy as a dependency

8.1.6 Migration Notes

PDF generation

If you wish to maintain the `trml2pdf` PDF generation (recommended if you have customized templates) for invoices, packing slips and shipping labels, you must set the `'DOCUMENT_CONVERTER'` key within `SATCHMO_SETTINGS` to `'shipping.views.TRMLDocument'`:

```
SATCHMO_SETTINGS = {
    ...
    'DOCUMENT_CONVERTER': 'shipping.views.TRMLDocument',
    ...
}
```

You should also move your templates, as the following template renames happened (here is indicated only the *template path*, from `templates/` onward):

- `shop/pdf/invoice.rml` → `shop/docs/rml/invoice.rml`
- `shop/pdf/packing-slip.rml` → `shop/docs/rml/packingslip.rml`
- `shop/pdf/shipping-label.rml` → `shop/docs/rml/shippinglabel.rml`

See *trml2pdf* for more details.

Sorl Thumbnail

In rev 2449 Satchmo is updated to use the latest Sorl Thumbnail code.

The primary change is that all custom templates that reference images, will need to be changed to support Sorl's updated syntax. A typical example of an old template tag looks like this:

```
{% thumbnail pic.picture 280x280 as image %}
<img src="{{ image }}" width="{{ image.width }}" height="{{ image.height }}" />
```

Templates need to be updated to support the endthumbnail tag:

```
{% thumbnail pic.picture "280x280" as image %}
<img src="{{ image.url }}" width="{{ image.width }}" height="{{ image.height }}" />
{% endthumbnail %}
```

Also, after upgrading to the latest Sorl Thumbnail code, you must remember to do a syncdb to create Sorl's cache table.

This upgrade allows Satchmo to utilize all the latest features in Sorl. You are encouraged to read Sorl's docs in order to determine which ones you'd like to include in your store.

Django Static Files

In rev 2476 Satchmo is updated to use Django's static files. If you have made any changes to templates, you'll need to make sure to replace references top media_url with STATIC_URL. Here is an example of the original template:

```
<script src="{{media_url}}js/jquery.cookie.js" type="text/javascript"></script>
```

Here is the new template:

```
<script src="{{ STATIC_URL }}js/jquery.cookie.js" type="text/javascript"></script>
```

8.2 Satchmo 0.9.2 Release

These are the notes for Satchmo 0.9.2. The full list of closed tickets can be viewed here

This release contains numerous fixes, improvements and updates from 0.9.1. A summary of some of the key items are included below.

8.2.1 New Features

- Display payment methods in emails

- Update the default templates to show quick orders and category indexes

- Ensure notes fields can be saved if they are included in the checkout process

- Display country names instead of abbreviations in templates

- Update google analytics code to use latest tracker js

- Add an unlimited downloads option for downloadable products

- Added support for Django's new CSRF support

- UPS Shipping module now allows you to box products together into 1 container

- Add a new config for the persistent cart to provide more options for how it handles returning users

- New template tag to show tiered shipping pricing table

- Control product variations inclusion in searching view the settings

- UPS module will display actual shipping times

- New template tag product_featured_list

- New config option to allow users to create their username instead of automatically generating

- Satchmo check now does a more thorough job of checking your environment for issues

- Product variation images can be shown when the user selects the product

- In the admin view of products, stock and price are displayed in a more useable format

- Ability to add and manage addressbook entries

8.2.2 Translations

- Added Dutch translation
- Added Czech translation
- Updated French translation
- Updated Spanish translation
- Added Romanian translation

8.2.3 Bug Fixes

- Fix issue if Unicode characters in options
- Fix issues with migrations and content types
- Resolve Paypal unneeded dependency on subscription product
- Make sure "all valid" discounts are correctly applied
- Fix Wishlish ajax add view
- Ensure that product variation properly inherits shipping from its parent
- Handling of gift certificate plus other payments fixed
- Bug in tax processor rendering fixed
- Bug in autosuccess payment module fixed
- Rebuild pricing was deleting prices if multiple sites were being used
- Currency template tags handles decimal places parameter correctly
- Improve robustness of South migrations
- Fix obscure issue where some users were able to edit the wrong information
- Remove custom read only widget use and use Django's newer read only field
- Paypal was not saving contact information correctly in certain cases
- Newsletter subscription status could be lost when checking out
- Use smart_attr for tiered weight shipping module
- Make sure that the product image file name doesn't exceed the length of the file field
- Multiple doc formatting fixes
- Authorize.net now uses shipee first and last names if given
- Typo in wishlist ajax view
- Make sure jquery correctly shows error messages on product pages
- Fixed problem where category caching was not handling translations correctly
- Ensure Ajax calls in checkout use SSL if it is turned on
- Akismet for comment moderation is fixed
- Improvements to South migrations
- Update Paypal IPN docs

- Fix issue where inactive or out of stock products could be purchased if there was a time lag between cart addition and checkout
- Cart template tag now allows show_discount
- Several fixes to transaction management
- Improve postgres syncdb support
- UPS time in transit prevents weekend pickups
- Payments weren't showing up correctly in admin
- Multiple updates to the tests to make them more robust
- Fix issue with persistent cart not working correctly
- Updates and fixes to sermepa payment module
- Fixes to syncdb
- Fedex module API changes
- Several fixes to handle unicode characters better
- Replace some admin jquery scripts with simpler django readonly fields
- Ensure that google checkout empties the cart
- Converted to SOAP based Fedex API
- Fix use of sale variable in context processor
- Handle different ship to and bill to addressees better
- Tax processor was not being cached correctly
- Secure autocomplete view so that only staff users could view it
- Fix product variation pricing so that explicit prices don't get overriden by deltas
- Fix apply_even_split so that discounts get applied properly
- Ensure home page paginator handles improper inputs better
- Improve satchmo_check's ability to report possible errors
- Fix payment information displayed in confirmation emails

8.2.4 Performance Improvements

- Tweaks to price queries to reduce unneeded queries
- Made a couple of view and template changes to reduce the number of SQL queries per page
- Added db index on Price Lookup table

8.2.5 Additional Packages

We have also made several updates to satchmo's dependencies including:

- livesettings
- keyedcache
- django-caching-app-plugins

You should upgrade these to the latest version when you upgrade your satchmo site.

8.2.6 Migration Notes

CSRF

As of rev 1999, Satchmo supports the newest Django 1.2 Cross Site Request Forgery (CSRF) protection. You are encouraged to read more about it int the Django docs

If you have an existing Satchmo store and need to support CSRF, you will need to do a couple of things:

1. Upgrade to Django 1.2.3 or greater

2. Ensure 'django.middleware.csrf.CsrfViewMiddleware' is in your MIDDLEWARE_CLASSES

3. Ensure that any forms in your custom templates, have {% csrf_token %} inside the <form> element

Messages

As of rev 1999, Satchmo incorporates some of the components of the new messages framework.

If you have an existing Satchmo store you will need to:

1. Add 'django.contrib.messages.middleware.MessageMiddleware' to your MIDDLEWARE_CLASSES

2. Add 'django.contrib.messages.context_processors.messages' to your TEMPLATE_CONTEXT_PROCESSORS

3. Add 'django.contrib.messages' to your INSTALLED_APPS setting

Protx Payment Module Renamed

As of rev 2051, the Protx module has been renamed to Sage Pay. You will need to manually copy over your former Protx credentials to the new Sage Pay module. No further changes should be necessary.

ProductPriceLookup Index

As of rev 2120, the price lookup model has an index on the productslug to improve performance on large data sets.

A south migration is included. To apply:

    python manage.py migrate product

Support for Django 1.3

As mentioned above, when using Django 1.3, it is important to upgrade django-caching-app-plugins (app_plugins) to version 0.1.2.

Active Products

Previously, category.active_products() would not consistently return the correct product variations. In rev 2168 this was fixed but had the unintended side effect of returning all the product variations in the default templates. In order to maintain consistency, the default variation display value is now False. The only people that will notice the issue are those that customized the variation display in the category page.

Order Shipping Method and Description Fields increased

The fields in the Order model for storing the shipping method and description were too small. The size of the field has been increased to 200 characters. A migration is included for this change:

> python manage.py migrate contact –fake 0001 python manage.py migrate shop –fake 0001 python manage.py migrate shop

Fedex Module Update

The old Fedex module used the XML api reference. The Fedex API documentation is geared more towards a SOAP API. We have moved to using this new interface. If you were using the old Fedex module, you will need to update your account and get a new API key.

Additionally, we are requiring two new modules for Fedex. You can install them using pip or easy_install:

> pip install fedex pip install suds

8.2.7 Additional Notes

Future versions of Satchmo will require Django 1.3.1 and higher.

8.3 Satchmo Release 0.9.1

Satchmo 0.9.1 includes many enhancements, updates, fixes and performance improvements over the prior release. All users starting new stores should use 0.9.1 as the basis for their stores.

A summary of some of the most notable changes are included below.

8.3.1 Changes

- Updated German translation
- Improvements to robustness of clonesatchmo program
- Template cleanups for gift certificates
- Improve discount handling for paypal
- Add new management command for billing subscription products
- Improve handling of ccv numbers
- Improve display string of subtypes in admin
- If language is chosen, make sure it's selected by default
- Tweak the product admin screens to show featured and active status
- Add new admin functions for toggling active/featured status of products
- Make the display of the site in the admin lists configurable through the settings
- Add a readonly field to the admin. Use it in the order admin for the status field
- Add some raw id fields in the admin so that large stores will work ok
- Fix the ajax functions to escape properly and use the right mime type

- Fix postgres typecast problem

- Added new Custom Product Attribute functionality - *Custom Product Attributes*

- Improved the code for calculating comment averages to increase performance

- Added signal for custom filtering of items which are subject to a discount

- Improve handling of the MEDIA_ROOT on Windows systems

- Add a documentation section about customizing the admin - *Customizing Admin*

- Handle error if unicode value is entered in the cart quantity field

- Add a persistent cart capability for previously logged in users

- Django 1.2 compatibility change in save signatures

- Remove some dupe code in the login functions

- Turn off autocomplete on cc#'s and ccvs in the checkout form

- Move the import of trml2pdf so that startup is quicker

- Fix a bug in the category slug url

- Change the way currency formats are determined. Explicit option will remove locale bugs.

- Fix the import for simplejson to support certain OS supplied json libs

- Update search to work on product variation skus

- Update login form so it accepts 75 characters

- Add a trap for CACHE_TIMEOUT

- Removed CATEGORY and PRODUCT slug configuration from livesettings

- Fixed imports in Canadapost shipping module

- Changed the way payment modules are imported. Making way for django-bursar integration

- Multiple fixes to paypal modules

- Allow admins to choose whether tax is calculated based on the shipping or billing address

- Refactor the templates and the way jquery is loaded. Will make it easier to switch jquery versions and hosting locations.

- Updated jquery to 1.4.2

- Added new documentation on Satchmo's views - *Satchmo Views*

- Cache the category tree for big performance improvements

- Refactor the email sending routine so it is more modular

- Add two new category tags

- Added search field in the admin to more easily find products

- Added new javascript to populate state fields when the country is changed

- Refactor some of the templates to make more consistent

- Change the behavior of applying discounts that don't evenly split among multiple products

- Update the autocomplete admin js to the latest version. Also widened field and added missing image

- Multiple documentation improvements on the shipping pages

- Added initial *South* migrations for the Product model - *Satchmo Migrations and Upgrades*

- Add optional support for sending HTML formatted emails

- Added Danish translation

- Performance improvement in the variation manager so that it won't choke on 1000's of products

- Added ability to assign discounts to an entire category - *Discounts*

- Expand product custom attributes to work with categories

- Refactored cart views for consistency with ajax calls

- Removed Keyedcache and Livesettings from Satchmo and gave them their own project

- Updated Russian, Polish, Italian, Portuguese and Spanish translations

- Added Hungarian translations

- Fix a bug in clonesatchmo

- Split out the product models to use a more modular layout and remove settings dependencies for loading

- Move SSL config to a setting, not a livesetting

- Fix bugs in payment authorization

- Improve error handling in Protx module

- Refactor portions of the ImageWithThumbnailField to be more generic

- Clarify an error page within the wishlist module

- Improve contact form handling if Country has no states

- Fix US SST tax module commands

- Update Brazilian country information

- Improved process for specifying file paths when using sendfile to deliver downloadable products

8.4 Satchmo 0.9 Release

This release is a major step forward for Satchmo. It includes a newly designed project layout meant to make it more modular and easier to integrate with the rest of your Django apps. This release also includes many bug fixes, improvements, new features and improved documentation. Some of the highlights are described below.

8.4.1 Changes

- Populate the xheaders so the admin bookmarklet will work

- New US SST tax module

- New Tiered weight shipping module

- Google checkout v2 integration

- New feature - satchmo toolbar which displays store info to the admins

- Added a new form_init signal - useful for collecting additional information at checkout

- Added a new object "OrderPaymentFailure" to track failures in the various payment processors

- New capability to enable and disable entire categories

- Added Canada Post shipping module

- Improved project layout to support easier integration with other Django apps

- Removed hard-coding of organizations types, organization roles and contact types from the contact app

- Added AUTH_EARLY option to Authorize.net

- Upgrade of Authorize.net SSL version to use ssle 3.0/tls

- Use sorl_thumbnail

- Allow products with the same slug to be on multiple sites

- Align the logout process with the Django conventions - allowing easier redirection

- Multiple improvements and fixes with discount code application

- Improvement to the installation process to make it simpler

- Fully compatible with Django 1.1

- Migrated to using Mercurial for version control

- Addition of new currency display in the Admin. Improved rounding support throughout

- Python 2.6 compatibility changes

- Improved several Ajax functions to be more consistent

- Addition of new SSL_PORT setting

- Improvements to the SSL Middleware to support more hosting setups

- Allow discount to be entered on first page of checkout and include free shipping

- Multiple improvements and cleanups in the templates to make inheritance easier

- Added improved jquery autocomplete code to the admin

- Several performance improvements in key code sections

8.4.2 Bug Fixes

- Fix inconsistent field length with gift certificates

- Fix many undefined variables found through pyflakes

- Fix IE javascript bug

- Fix several infinite recurison edge cases

- Improve error handling if large quantities are entered when buying product

- Fix Paypal IPN callback url

8.5 Satchmo 0.8.1 Release

Welcome to Satchmo 0.8.1. This release is a minor bug fix release after 0.8. The main purpose of this release is to provide a stable checkpoint before some backwards incompatible (but very useful) reorganization is going to hit trunk.

8.5.1 Changes

- Fix user name generator to handle unicode chars more robustly

- Fix handling of ampersands in PDF generation

- Update Italian translation

- Update DOB widget

- Added neve cache decorator to several views

- Code speedups for discount checking

8.6 Satchmo 0.8 Release

Welcome to Satchmo 0.8. This release includes many features, enhancements and bug fixes.

8.6.1 New Features

Satchmo Trunk includes many new features and improvements. Some of the highlights are:

- New most popular, best sellers and recently added views

- New tiered shipping module based on quantity

- New Fedex shipping module

- New USPS shipping module

- New tiered shipping module

- New optional app "satchmo.upsell" which allows product upselling on product detail pages

- New signals in get_qty_price methods of products, allowing code to manipulate the price returned by the function. Example code using these signals is at satchmo.contrib.cartqty_discounts, which modifies the qty_discount method to calculate the discount based on the total number of items in the cart, rather than the total amount of the lineitem.

- Improved inheritance of detail_producttype.html pages, which no longer need to duplicate as much of the code of base_product.html, allowing for easier addition of features to the product detail pages.

- Migrated to newforms admin which fixed some bugs and provides a much more solid base for future changes

- Added multi-shop capability

- Multiple price tier support

- Option to require login before checkout

- Most popular, best sellers and recently added views

The following documentation improvements were made:

- Addition of upsell documentation

- Improved dependency installation instructions

- Multiple fixes and clarifications

- Additional documentation on signals

The following payment modules were added or updated:

- New Protx payment module

- Purchase order module

- Authorize.net supports recurring payments

The following translations were updated or added:

- German

- French

- Hebrew

- Turkish

- Italian

8.6.2 Backwards-incompatible Changes

This release does introduce some backwards incompatible-changes so you are encouraged to review and understand the changes listed here

8.6.3 Thanks

To everyone in the Satchmo community for providing feedback, patches and support.

8.7 Satchmo 0.7 Release

Welcome to Satchmo 0.7. This release occurs after nearly 7 months of additional development and includes major new features, bug fixes and lots of other improvements. Anyone working on a Satchmo project is encouraged to use this release as the baseline for their store.

8.7.1 New Features

Satchmo 0.7 includes many new features and improvements. Some of the highlights are:

- Capability to choose language translation

- Improvements to the category code and display including adding images to categories

- New translation capabilities were added for all parts of Satchmo

- Improved Custom Product which supports much more broad usage

- Currency is displayed based on the current locale

- Allow featured products to be randomly displayed

- Increase phone number field length to support more country's phone numbers

- Updated google analytics to use new google code but allow fall back to old urchin.js

- New template tags: product_category_siblings and product_images

- New tax module to calculate taxes based on country or state

- Updated discount system to track which discounts were applied to which line items.

- Add weight and dimension unit info to the product model.

- Created integration with UPS online tools
- Created new Shipping Module, 'Tiered', allowing multiple carriers and variable pricing by cart total.
- Added sku model field
- Added gift certificate functionality
- Cleaned up New Zealand l10n data
- Improved tax calc code
- Improved security by ensuring dupe emails couldn't be used
- Fixed PDF generation bug in windows
- Added support for categories in the product export function
- New subscription product with recurring billing support
- CSS fixes and improvements to breadcrumbs and other parts of base store
- Remove storage of ccv field from database
- More control for purchasing items not in stock
- New recently viewed items support
- Added wishlist functionality
- UI improvements on the admin site
- Disable shipping address fields if user selects "same shipping and billing address are used"
- SEO optimizations to templates
- Massive performance improvements when using Configurable Products
- Unicode fixes to shipping and payment backends
- Allow PDF logo to be selected from the admin site
- Added Google Feed support
- Support for store wide sales via "auto discounts"

The following documentation improvements were made:

- Documentation system now uses Sphinx
- New installation instructions
- Improved documentation for custom payment modules

The following payment modules were added or updated:

- New Cybersource module
- New COD module
- Bug fixes in Trustcommerce

The following translations were updated or added:

- German
- Swedish
- Brazilian Portuguese
- Bulgarian

- Korean

- French

8.7.2 Backwards-incompatible Changes

This release does introduce some backwards incompatible-changes so you are encouraged to review and understand the changes listed here

8.7.3 Thanks

Thanks to the many people who have contributed to or supported the project. This release includes some new translations and we are very appreciate of those who have taken the time to contribute their linguistic skills to the project.

A special thanks goes to Bruce Kroeze who continues to provide major improvements and fixes to Satchmo. The project would not be where it is today without his support.

8.8 Satchmo Migrations and Upgrades

The latest list of backwards incompatible changes can always be found here-http://bitbucket.org/chris1610/satchmo/wiki/BackwardsIncompatibleChanges

The ideal way to migrate would be to dump all of your existing store data, remove all of your old tables, synch the new models and reload the data. If this process is not practical, then follow the individual steps outlined below.

Always remember to do a complete backup of your store before attempting to migrate. Additionally, we recommend that you test the migration on a test server before attempting on a production server.

8.8.1 Using South

The very latest version of Satchmo includes migration files for South. So far, we only have migrations for the `product` app.

The instructions below assume that you have installed South 0.7; you can download it from here:

> http://south.aeracode.org/docs/installation.html

Note: The initial product migration is based on the 0.9 code base. If you already have an existing installation, please read Upgrading from an existing 0.9 installation; if you are planning to do a fresh installations of 0.9.1, please see Installing Satchmo 0.9.1 with South.

Upgrading from an existing 0.9 installation

- Download/check out a copy of the new version of Satchmo.

  Warning: *Do not* install the new version of Satchmo yet.

- If you aren't already using South for your project, add it to your `INSTALLED_APPS` and create the history table:

```
$ python manage.py syncdb

Syncing...
Installing yaml fixture 'initial_data' from '/home/user/lib/python2.5/Satchmo-0.9_0-py2.5.egg/sa
Installing yaml fixture 'initial_data' from '/home/user/lib/python2.5/Satchmo-0.9_0-py2.5.egg/sa
Installing yaml fixture 'initial_data' from '/home/user/lib/python2.5/Satchmo-0.9_0-py2.5.egg/pr
Installed 16 object(s) from 3 fixture(s)

Synced:
 > django.contrib.admin
 > django.contrib.auth
 > django.contrib.contenttypes
 > django.contrib.sessions
 > django.contrib.sites
 > django.contrib.comments
 > satchmo_store.shop
 > keyedcache
 > livesettings
 > l10n
 > south
 > sorl.thumbnail
 > satchmo_store.contact
 > tax
 > tax.modules.area
 > tax.modules.percent
 > shipping
 > product
 > payment
 > payment.modules.giftcertificate
 > satchmo_utils
 > app_plugins
 > singpost
 > shop_ext
 > satchmo_state

Not synced (use migrations):
 -
(use ./manage.py migrate to migrate these)
```

- Replace your Satchmo installation with the new version of it.

- Since the database tables have already been created for your old (0.9) Satchmo installation, you need to "fake" the initial migration, or else South will try re-creating the database tables. Do it like so:

```
$ python manage.py migrate product --fake 0001

- Soft matched migration 0001 to 0001_initial.

Running migrations for product:
 - Migrating forwards to 0001_initial.
 > product: 0001_initial
   (faked)
```

- See list of migrations:

```
$ python manage.py migrate product --list

product
  (*) 0001_initial
```

```
( )  0002_add_attributeoption
( )  0003_add_productattribute_option
( )  0004_remove_productattribute_name
( )  0005_fix_attributeoption_error_default_spelling
( )  0006_custom_textfield_add_constraint
( )  0007_add_discount_valid_products_field
( )  0008_remove_discount_validproducts_field
( )  0009_add_categoryattributes
```

- Run the migrations:

```
$ python manage.py migrate product --db-dry-run # db untouched
$ python manage.py migrate product
```

Installing Satchmo 0.9.1 with South

Note: We asume that you've already installed South, but have yet to take the leap with Satchmo; see http://south.aeracode.org/docs/installation.html for details.

- Before installing Satchmo, ensure that your tables and migrations are up-to-date, by running `syncdb` and `migrate`.

- Install Satchmo by adding it to your path and `INSTALLED_APPS`.

- You have two options:

  - Create the database tables for Satchmo's non-migration models, then create the `product` app tables and run the migrations:

    ```
    $ python manage.py syncdb
    $ python manage.py migrate product
    ```

  - Alternatively, you can create the tables directly and skip running the migrations:

    ```
    $ python manage.py syncdb --all
    $ python manage.py migrate product --fake
    ```

Note: Users familiar with the codebase may wonder why we didn't tell South to apply migrations for the product modules, like `configurable` and `custom`, even though they are there; that's because they have already been implicitly applied. You can verify this with an installation-wide migration listing:

```
$ python manage.py migrate --list

 product
  (*)  0001_initial
  (*)  0002_add_attributeoption
  (*)  0003_add_productattribute_option
  (*)  0004_remove_productattribute_name
  (*)  0005_fix_attributeoption_error_default_spelling
  (*)  0006_custom_textfield_add_constraint
  (*)  0007_add_discount_valid_products_field
  (*)  0008_remove_discount_validproducts_field
  (*)  0009_add_categoryattributes
  (*)  0010_add_discountable_categories
  (*)  0011_split_products
  (*)  0012_update_contenttypes
```

```
configurable
 (*) 0001_split
 (*) 0002_update_contenttypes

custom
 (*) 0001_split
 (*) 0002_update_contenttypes

downloadable
 (*) 0001_split
 (*) 0002_update_contenttypes

subscription
 (*) 0001_split
 (*) 0002_update_contenttypes
```