

Satchmo

The webshop for perfectionists with deadlines.



**Djangocon Presentation
Bruce Kroeze & Chris Moffitt
September 6th, 2008**

Discussion Topics

- Introduction and history
- Unique Features
- Large Project Issues
- The Future



Satchmo - The Beginning

Excerpt from the initial post:

"I know some people have started (or have completed) work on their own stores. Is there any framework out there that I can leverage or do I have to "roll my own?" I think it would be a great thing for this community to create a store framework. The Django pluses I mentioned would be awesome in a complete solution. I think it would also attract a lot of users to the project. Is there interest or any activity in this area? I'm willing to hack through by myself but I think if there's anyone that is an expert, we could move through this a lot more quickly."

- Django Users Posting by Chris Moffitt April 12, 2006

What's in a name?



Googling for "'too commercial' jazz' turns this up:

<http://www.redhotjazz.com/louie.html>

It's Louie "Satchmo" Armstrong. I could go for Armstrong, but I think "Satchmo" is cooler.

- jmj

Posted by Jeremy Jones on April 14th, 2006

What is Satchmo?

- Django-based framework for developing unique and highly customized ecommerce sites
- Designed for *developers* that have a unique store need
- Supported by over 570 people in Satchmo-users Google Group
- Deployed to 9 known production sites & more in development
- Translated into multiple languages
- Preparing for it's 1.0 release!

Satchmo Features

- Template driven
- Multiple payment gateways
- Custom shipping modules
- Custom payment modules
- UPS, FedEx integration
- Downloadable, subscription & custom products
- Gift certificates
- Tax & VAT
- Multiple shops-per-instance
- Multiple discount options
- Wishlists
- Full account management options
- PDF invoice & shipping documents
- Automatic image thumbnails
- Modular newsletters
- Strong internationalization and localization

And Much [More](#) ...

Components

Applications (core)

'satchmo.caching',
'satchmo.configuration',
'satchmo.shop',
'satchmo.contact',
'satchmo.product',
'satchmo.shipping',
'satchmo.payment',
'satchmo.discount',
'satchmo.supplier',
'satchmo.thumbnail',
'satchmo.l10n',
'satchmo.tax',
'satchmo.productrating'

Middleware

- "threaded_multihost.middleware.ThreadLocalMiddleware",
"satchmo.shop.SSLMiddleware.SSLRedirect",
"satchmo.recentlist.middleware.RecentProductMiddleware"

Context Processors

- 'satchmo.recentlist.context_processors.recent_products',
'satchmo.shop.context_processors.settings',

Applications (non-core)

'satchmo.shipping.modules.tiered',
'satchmo.newsletter',
'satchmo.feeds',
'satchmo.giftcertificate',
'satchmo.recentlist',
'satchmo.wishlist',
'satchmo.upsell',

Unique Features - Custom Shipping

Satchmo supports many shipping modules but the real power is in the flexibility to create your own.



Scenario - You have a small retail presence and would like to allow people within 20 miles of your store to stop by and pick up the purchase instead of spending money for shipping

Solution - Create your own shipping module to determine how far the address is from your shipping point.

Bonus – Use Google Map's API to do it!

Creating a Custom Shipping Module

Steps to create a your "local pickup" shipping module:

1. Make a new directory somewhere. Ex: `mysite.localpickup`
2. Copy in the boilerplate from `satchmo.shipping.modules.per.`
3. Only two files of interest: `config.py` and `shipper.py`
- ε. `config.py` uses another Satchmo feature, the configuration system, and defines the custom variables used by this shipper.
0. `shipper.py` uses a standard "walks like a duck" Python interface for handling shipping calculations.

Local Pickup Shipper Code - Boilerplate

```
from django.utils.translation import ugettext, ugettext_lazy
from satchmo.configuration import config_value
_ = ugettext_lazy
from satchmo.shipping.modules.base import BaseShipper
from geopy import geocoders
from geopy import distance
from satchmo.shop.models import Config
```

```
class Shipper(BaseShipper):
    id = "LocalPickup"
```

```
    def __str__(self):
```

```
        """
```

```
        This is mainly helpful for debugging purposes
```

```
        """
```

```
        return "Local Pickup"
```

```
    def description(self):
```

```
        """
```

```
        A basic description that will be displayed to the user when selecting their shipping options
```

```
        """
```

```
        return _("Local Pickup")
```

```
    def cost(self):
```

```
        """
```

```
        Complex calculations can be done here as long as the return value is a dollar figure
```

```
        """
```

```
        fee = config_value('SHIPPING', 'LOCAL_PICKUP_FEE')
```

```
        return fee
```

```
    def method(self):
```

```
        """
```

```
        Describes the actual delivery service (Mail, FedEx, DHL, UPS, etc)
```

```
        """
```

```
        return _("Local Pickup")
```

Local Pickup Shipper Code - Workhorse

```
def expectedDelivery(self):
```

```
    """
```

```
    Can be a plain string or complex calculation returning an actual date
```

```
    """
```

```
    return _("5 Hours")
```

```
def valid(self):
```

```
    """
```

```
    Can do complex validation about whether or not this option is valid.
```

```
    """
```

```
    return self._destination_is_local()
```

```
def _destination_is_local(self):
```

```
    store_config = Config.objects.get_current()
```

```
    store_add = u"%s, %s" % (store_config.street1, store_config.postal_code)
```

```
    customer_add = u"%s, %s" % (self.contact.billing_address.street1,  
                                self.contact.billing_address.postal_code)
```

```
    api_key = config_value('SHIPPING', 'GOOGLE_API_KEY')
```

```
    g = geocoders.Google(api_key)
```

```
    _, store = g.geocode(store_add)
```

```
    _, dest = g.geocode(customer_add)
```

```
    dist = distance.distance(store, dest).miles
```

```
    return(dist < config_value('SHIPPING', 'LOCAL_PICKUP_DISTANCE'))
```

Local Pickup Config Code

```
from django.utils.translation import ugettext_lazy as _
from satchmo.configuration import *
SHIP_MODULES = config_get('SHIPPING', 'MODULES')
SHIP_MODULES.add_choice(('satchmo.shipping.modules.localpickup', _('Local Pickup')))

SHIPPING_GROUP = config_get_group('SHIPPING')
config_register_list(

    DecimalValue(SHIPPING_GROUP,
        'LOCAL_PICKUP_FEE',
        description= _("Pickup Fee"),
        requires=SHIP_MODULES,
        requiresvalue='satchmo.shipping.modules.localpickup',
        default="5.00"),

    DecimalValue(SHIPPING_GROUP,
        'LOCAL_PICKUP_DISTANCE',
        description= _("Radius for local pickup"),
        requires=SHIP_MODULES,
        requiresvalue='satchmo.shipping.modules.localpickup',
        default="10.0"),

    StringValue(SHIPPING_GROUP,
        'GOOGLE_API_KEY',
        description= _("Google Maps API Key"),
        help_text= _("Google API Key"),
        requires=SHIP_MODULES,
        requiresvalue='satchmo.shipping.modules.localpickup',
        default=""),

)
```

Enabling the Local Pickup Module

In the site's settings.py, we have a dictionary for Satchmo settings. Add the module as an option there, and it will get picked up as an allowable shipper.

```
SATCHMO_SETTINGS = {  
    # ... skipping the other settings here  
    CUSTOM_SHIPPING_MODULES = ['mysite.localpickup'] }
```

Then, you would activate the newly built shipping module at your site settings page usually “mysite.com/settings/”

That's also where you would configure:

- Maximum distance
- Google API key
- Fee

Configuring the Local Pickup Module

Shipping Settings (Hide)	
<p>Active shipping modules Select the active shipping modules, save and reload to set any module-specific shipping settings. Default value: Per piece</p>	<p>Per piece FEDEX Flat rate UPS Local Pickup</p>
<p>Google Maps API Key Google API Key Default value: ""</p>	<input type="text"/>
<p>Pickup Fee Default value: 5.00</p>	<input type="text" value="5.00"/>
<p>Radius for local pickup Default value: 10.0</p>	<input type="text" value="10.0"/>
Tax Settings (Show)	

Unique Feature - Configuration

Why?

- Updating configuration files isn't always practical
- We got tired of always having to make backwards incompatible notes just to add a new feature
- The Shop object got to be a dumping ground and it was a hassle.
- Configuration app is aware of prerequisites, so you don't see sections which aren't applicable to you. No UPS on your store, then no need to have all that data in the db.
- Very fast, heavily cached access to all config variables.

Unique Feature - Caching

Why?

- Able to cache a None object, and know that they weren't cache misses.
- A consistent design patterns for cache application.
- Caching stats and resets from the app itself.

Unique Feature – Payment Modules



- Satchmo's payment module design supports a broad range of payment processors:
 - URL posting and response - **Authorize.net**
 - Full XML message creation and response – **Cybersource**
 - Utilize full power of Django templates for XML message creation
 - Elementree for simple parsing of data
 - Complex site redirection/posting – **Google Checkout, Paypal**
 - Uses custom views to manage the process
 - Custom interface via libraries – **TrustCommerce** via Tclick

Signal Usage in Satchmo



We currently have 13 signals in the base Satchmo.

"satchmo_order_success" and "satchmo_cart_changed" are the two most heavily used.

The use of signals allowed us to remove circular dependencies, clear up responsibilities, and make the whole thing much more understandable.

We'll be adding many more as we grow to support better integration.

Real-World Signal Usage

Loyalty Card Creation



Scenario: Develop a site for a customer who wishes to sell “loyalty cards” which offer a 10% discount, each have a unique code, and automatically expire in 1 year.

Approach: Listen for “order_success” signal, look through cart for loyalty cards, create a unique discount code for each one found, email the code to the user and to the fulfillment house, update order notes with loyalty card code.

Total time to implement with testing: 1.5 hours.

Real-World Signal Usage

Custom Tiered Pricing

Scenario: Develop a site for a customer who has two classes of users, each of which see and pay different prices for products.



Approach: Listen for “satchmo_price_query” signal. Look up user from threadlocals. If user is part of special group, look up discount for that group, apply to current price, and return modified price.

Total time to implement with testing: 2 hours.

Integrating with other Django Apps - Banjo

Scenario: Customer wants to have a Blog integrated with their store.



Django App Used: Banjo

Interesting bits:

- Blog entries can be integrated onto the homepage
- Entries can be tagged so that they show related products
- Blog remote-posting interface can be used to edit/maintain pages instead of the manual “flatpages” app.

Steps to Integrate Banjo

- ✂ Add requirements to `INSTALLED_APPS` and `CONTEXT_PROCESSORS`
- ✂ Make a custom templatetag library which looks up related products using the tags attached to the blog entry. Put this in the local “site” app.
- ✂ Override the Satchmo `base_index.html` template and the Banjo `blog/view_post.html` template.



Big vs. Small

The Endless Debate

Should Satchmo be:

- A single app
- Focused on small stores
- Capable of supporting thousands of products
- Easily integrated with other “Enterprise” apps
- Heavily “instrumented” for reporting that some stores need



Big vs. Small

The Working Philosophy



Satchmo will remain usable standalone.
Core development is driven by real sites.

We appreciate contributions providing new functionality if they:

- have tests
- have documentation (sadly rare)
- are optional
- don't require modification of basic functionality or addition of special cases
- need new signals (we'll add them if they make sense)

It is simply amazing how far signals + templatetags will get you.

Documentation

We strive to emulate Django's documentation example.

ReStructured text for all the documents

Sphinx for presenting them nicely

Use built in admin docs functionality

The biggest challenge is providing the right level of documentation to help newbies as well as experienced devs



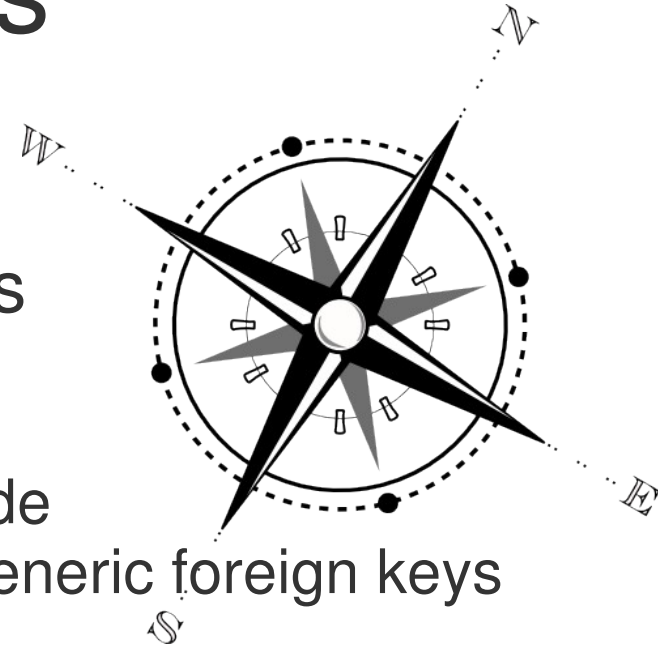
The Future

- Stabilizing on Django 1.0
 - YEAH!
- Improving the documentation around extensibility
- Improved product model
- More customization of the admin interface



The Future - Product Models

- Current product model has some limitations
 - Not straightforward to extend
 - Tend to proliferate a lot of different model types
 - Lots of special case checking throughout the code
 - Doesn't take full advantage of inheritance and generic foreign keys
- Future model
 - Create a design that is easily extensible for unique shops
 - Clean interface to other sections of the code
 - Support just about any product that could be sold on the web



The Future - Admin



New Admin Features in the works:

Order Manager:

- Multi-tab interface
- Modify orders, adjust discounts, add notes, set status, add order items, change quantities, mark manual payments

Product Manager:

- Built to handle thousands of products & variations
- Integrated assignment of categories
- Importing and Exporting of product lists to common formats

Reporting – we need some!

The Future - External Apps



XMLRPC interface

- Use Satchmo as backend to Flash store
- Desktop product manager
- Desktop reporting

JSON interface

- AIR/Google/Yahoo/OSX store status widgets

Questions ?