django-oscar-mws Documentation

Release 0.1.0

Sebastian Vetter

January 08, 2017

Contents

django-oscar-mws is still under heavy development and things are changing quickly. That means the few pieces of documentation currently available are likely to change or might even be obsolete. It also explains why the docs are pretty much non-existent. Stay tune, I'll try and improve them as I go along.

Contents:

CHAPTER 1

Concepts

django-oscar-mws (OMWS) provides a few models that represent data retrieved from or sent to Amazon's MWS API.

1.1 Merchant Account

- The merchant account represents the overall account for a region such as EU, US.
- A merchant account has to be linked to a stock record to be able to store stock for a given product in the right place. A merchant has a 1-to-1 relationship to the `partner.Partner` model.
- When saving a merchant account without a partner, a partner with name `Amazon (<MWS_REGION>)` is looked up or created with the merchant's region corresponding to `MWS_REGION`. E.g. for a US merchant account this would be `Amazon (US)`.

1.2 Stock Records with MWS

Using MWS for fulfillment implies that we are handling physical stock that requires shipping and the tracking of stock. Oscar's `StockRecord` model provides all the necessary functionality for this. However, there is a couple of assumptions that we have to make based on the way MWS works.

1. Stock in MWS is available on the *merchant account* level which can be mapped to a fulfillment region, e.g. Europe. As a result, we have to handle one stock record per region/seller account which is done by tying a `MerchantAccount` directly to a `Partner`. This is automatically taken care of when saving a new merchant account.
2. Oscar, by default, tracks stock and uses a 2-stage approach for it. The amount of stock is stored in `num_in_stock`. Whenever a customer successfully places an order for an item, the `num_allocated` on its stock record is incremented. The actual amount that is available to buy is calculated by subtracting the allocated stock from the number in stock:

   ```
   available = stockrecord.num_in_stock - stockrecord.num_allocated
   ```

   This makes tracking stock from MWS a little tricky because we can't just set the `num_in_stock` value to the supply quantity retrieved from MWS. This would ignore the allocated stock number and result in a wrong number of items available to buy. Resetting `num_allocated` to zero when updating inventory will cause issues by itself because marking an item as shipped will result in decrementing `num_in_stock` and `num_allocated` by the shipped quantity which would also result in wrong stock numbers. We decided for a combined solution by resetting `num_allocated` to zero when updating stock from MWS and then preventing decrementing stock when it is marked as shipped if the stock record is tracking MWS stock. This functionality is encapsulated in `AmazonStockRecordMixin` which you should add to your projects `StockRecord`.

CHAPTER 2

Getting Started

2.1 Setting Up The Sandbox

django-oscar-mws comes with a sandbox site that shows how MWS can be integrated with Oscar. It resembles a basic set up of Oscar with an out-of-the-box integration of MWS. This section will walk you through setting the sandbox up locally and how to make it interact with the MWS API.

Note: Oscar itself has quite a few dependencies and settings that might cause some problems when you are setting up the MWS sandbox. In addition to this documentation you might also want to check out the Oscar docs on setting up a project.

The first thing to do is cloning the repository and installing it's requirements which will includes setting up Oscar. It also creates a new database (if it doesn't exist) creates the required tables:

```
$ git clone git@github.com:tangentlabs/django-oscar-mws.git
$ cd django-oscar-mws
$ mkvirtualenv mws  # requires virtualenvwrapper to be installed
$ make sandbox
```

By default, the sandbox is using Oscar's precompiled *CSS* files by setting `USE_LESS = False`. If you want to use *LESS* to generate the CSS yourself, take a look at the documentation on how to use LESS with Oscar.

2.1.1 Create Admin User

The main interface for MWS lives in Oscar's dashboard and therefore requires an admin user to login. Create a new admin account using Django's `createsuperuser` command and follow the instruction:

```
$ ./sandbox/manage.py createsuperuser
```

You should now be able to run the sandbox locally using Django's builtin HTTP server:

```
$ ./sandbox/manage.py runserver
```

You now have a sample shop up and running and should be able to navigate to the dashboard to continue the setup of your MWS credentials.

2.1.2 Stock Records and MWS

As described in *Concepts*, integration Oscar's stock records with MWS requires a little additional setup. Oscar assumes that it handles the allocation and consumption of stock through the stock record(s) for a product. With MWS the available stock is actually dictated by Amazon and can't be handled the Oscar way. Therefore, a few extra methods on the stock record are required which are encapsulated in the `AmazonStockTrackingMixin`.

Making these methods available to OMWS requires you to override the `partner` app in Oscar. Check the documentation on how to customise Oscar apps to get a more comprehensive introduction. The short version is, you need to create a new app in your project called `partner` and create a `models.py` module in it. Import all the models from the core Oscar app and add the `AmazonStockTrackingMixin` to the `StockRecord` model similar to this:

```
from oscar.apps.address.abstract_models import AbstractPartnerAddress
from oscar.apps.partner.abstract_models import *

from oscar_mws.mixins import AmazonStockTrackingMixin


class StockRecord(AmazonStockTrackingMixin, AbstractStockRecord):
    pass


class Partner(AbstractPartner):
    pass


class PartnerAddress(AbstractPartnerAddress):
    pass


class StockAlert(AbstractStockAlert):
    pass
```

And then add the `partner` app to your `INSTALLED_APPS` like this:

```
from oscar.core import get_core_apps

INSTALLED_APPS = [
    ...
] + get_core_apps(['myproject.partner'])
```

This setup provides you with a default implementation that disables updating the consumed stock on a MWS-enabled stock record and provides methods to update stock from MWS when retrieved from Amazon.

Note: The `AmazonStockTrackingMixin` provides a basic implementation for MWS-enabled stock. If you are using multiple different types of fulfillment partners this implementation might not be sufficient and you'll have to adjust the implemenation to your specific use cases.

2.2 Setting Up MWS

The API endpoints provided by Amazon MWS differ based on the MWS region. The different regions and endpoints are detailed in the Amazon docs. Each region requires separate MWS credentials for each account. In OMWS, these accounts are called *merchant accounts* and are used to identify the endpoints to use when communication with MWS.

You have to create a merchant account and provide your MWS credentials to be able to connect to MWS. Head to the *Amazon MWS > Merchants & Marketplaces* in the Oscar dashboard and select 'Add merchant account'. A corresponding partner account in Oscar is required for a MWS merchant account, however, if no partner is selected explicitly, a new one will be created automatically with the same name as the MWS merchant account.

With your merchant account(s) added, you can update the corresponding marketplaces in the drop-down menu on the right-hand side. This will pull the MWS marketplaces that you are able to trade in from MWS. This will also indicate that communicating with the MWS API is successful.

CHAPTER 3

Settings

3.1 MWS_ENFORCE_PARTNER_SKU

default: `True`

The seller SKU for a product used with Amazon to uniquely identify it stored on the `AmazonProfile` of that product. Oscar's stock record in the *partner* app also provides a SKU that is used with a *Partner* corresponding to a seller/merchant ID with MWS. In most cases, you would want the partner SKU on the `StockRecord` kept in sync with the *SKU* on the `AmazonProfile`. To enforce this constraint, you can update the stock records for Amazon-related partners whenever the Aamzon profile is saved. This is enabled by default. To switch it off set `MWS_ENFORCE_PARTNER_SKU = False` in you settings.

3.2 MWS_ORDER_ADAPTER

Specify the order adapter class to use to convert an order into a fulfillment order containing data as expected by Amazon.

3.3 MWS_ORDER_LINE_ADAPTER

The mapper class for the order line to convert it into a fulfillment orde line including data as expected by Amazon.

3.4 MWS_FULFILLMENT_MERCHANT_FINDER

default: `oscar_mws.fulfillment.finders.default_merchant_finder`

3.5 MWS_DEFAULT_SHIPPING_SPEED

default: `Standard`

CHAPTER 4

Recipes For Commmon Problems

Sorry but you'll need to be a little patient. I'll get to it as soon as possible.

CHAPTER 5

Notes

Warning: For same parts of the API to work, you'll have to provide tax information in your MWS Pro account. Otherwise you'll get a `Seller is not registered for Basic fulfillment.` error message back.

For the time being, this is going to be a collection of finding while using the MWS API. It mainly things that I've picked up while working on it through feedback submitting wrong or incomplete data. It's not necessarily correct and I am happy to be corrected where that's the case.

5.1 Fulfillment

- Fulfillment orders are created against a seller account rather than a marektplace. That means all marketplaces that belong to the same seller account are submitted against that seller account and do not require a marketplaces.

5.1.1 Submitting An Order

- The `DestinationAddress.CountryCode` is validated against the seller account region and is rejected if outside of it. E.g. a `US` country code submitted to a seller acount for Europe is rejected with:

```
<Error>
  <Type>Sender</Type>
  <Code>InvalidRequestException</Code>
  <Message>Value US for parameter DestinationAddress.CountryCode is invalid. Reason: InvalidValu
</Error>
```

- Submitting an order requires a value for `StateOrProvinceCode` for the destination address. As far as I have tested it, there is no validation on the state for the European marketplaces. The Marketplace for the US (and most likely Canada as well) is rejecting anything but the official 2-letter code for the US state.

CHAPTER 6

API Reference

6.1 Models and Mixins

class `oscar_mws.mixins.`**`AmazonStockTrackingMixin`**

A mixin to make stock tracking for Amazon MWS fulfilled products possible. The way stock tracking works in Oscar doesn't play nicely with the details returned from MWS. Basically Amazon provides a single value which is the amount of items still available to be fulfilled. In Oscar, we track the number in stock as well as the allocated number of products. `num_in_stock - num_allocated` is the number of items actually avaiable to buy and both number in stock and number allocated are only decremented whenever an item is marked as shipped.

To handle this properly and be able to synchronise the fulfillable number of products available from Amazon, we use this mixin to override the

`consume_allocation`(*quantity*)

This is used when an item is shipped. We remove the original allocation and adjust the number in stock accordingly

Parameters **quantity** (*integer*) – The quantity to be consumed.

`is_mws_record`

Checks whether this stock record is associated with an Amazon merchant account.

Rtype bool `True` if the stockrecord is Amazon stock, `False` otherwise.

`set_amazon_supply_quantity`(*quantity*, *commit=True*)

Convenience method to set the field `num_in_stock` to *quantity* and reset the allocated stock in `num_allocated` to zero. We don't care about allocation for MWS stock and therefore just reset it.

Parameters

- **quantity** (*integer*) – The quantity currently available on Amazon for Fulfillment by Amazon (FBA).
- **commit** (*boolean*) – Allows to prevent immediate saving of the changes to the database. This is useful if you want to save on database queries when making other changes to the stock record.

6.2 Feeds

6.3 Fulfillment

CHAPTER 7

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index

Index