dj-stripe Documentation

Release 1.0.0.post1

Alexander Kavanaugh

Nov 19, 2017

Contents

- Subscription management
- Designed for easy implementation of post-registration subscription forms
- Single-unit purchases
- Works with Django >= 1.10
- Works with Python 3.6, 3.5, 3.4, 2.7
- Built-in migrations
- Dead-Easy installation
- Leverages the best of the 3rd party Django package ecosystem
- *djstripe* namespace so you can have more than one payments related app
- Documented
- 100% Tested
- Current API version (2017-06-05), in progress of being updated

CHAPTER 1

Contents

1.1 Installation

1.1.1 Get the distribution

At the command line:

```
$ pip install dj-stripe
```

1.1.2 Configuration

Add `djstripe` to your `INSTALLED_APPS`:

```
INSTALLED_APPS += [
    'django.contrib.sites',
    # ...,
    "djstripe",
]
```

Add your Stripe keys and set the operating mode:

```
STRIPE_LIVE_PUBLIC_KEY = os.environ.get("STRIPE_LIVE_PUBLIC_KEY", "<your publishable␣
↪key>")
STRIPE_LIVE_SECRET_KEY = os.environ.get("STRIPE_LIVE_SECRET_KEY", "<your secret key>")
STRIPE_TEST_PUBLIC_KEY = os.environ.get("STRIPE_TEST_PUBLIC_KEY", "<your publishable␣
↪key>")
STRIPE_TEST_SECRET_KEY = os.environ.get("STRIPE_TEST_SECRET_KEY", "<your secret key>")
STRIPE_LIVE_MODE = <True or False>
```

Add some payment plans via the Stripe.com dashboard or the django ORM.

Add the following to the *urlpatterns* in your *urls.py* to expose the webhook endpoint:

```
url(r'^payments/', include('djstripe.urls', namespace="djstripe")),
```

Run the commands:

```
python manage.py migrate

python manage.py djstripe_init_customers
```

1.1.3 Running Tests

Assuming the tests are run against PostgreSQL:

```
createdb djstripe
pip install -r tests/requirements.txt
tox
```

1.2 Usage

Nearly every project breaks payment types into two broad categories, and will support either or both:

1. Ongoing Subscriptions (Well supported)
2. Individual Checkouts (Early, undocumented support)

1.2.1 Ongoing Subscriptions

dj-stripe provides three methods to support ongoing subscriptions:

- Middleware approach to constrain entire projects easily.
- Class-Based View mixin to constrain individual views.
- View decoration to constrain Function-based views.

> **Warning:** **anonymous** users always raise a `ImproperlyConfigured` exception.
>
> When **anonymous** users encounter these components they will raise a `django.core.exceptions.ImproperlyConfigured` exception. This is done because dj-stripe is not an authentication system, so it does a hard error to make it easier for you to catch where content may not be behind authentication systems.

Any project can use one or more of these methods to control access.

Constraining Entire Sites

If you want to quickly constrain an entire site, the `djstripe.middleware.SubscriptionPaymentMiddleware` middleware does the following to user requests:

- **authenticated** users are redirected to `djstripe.views.SubscribeFormView` unless they:
  - have a valid subscription –or–
  - are superusers (user.is_superuser==True) –or–

  - are staff members (user.is_staff==True).

- **anonymous** users always raise a `django.core.exceptions.ImproperlyConfigured` exception when they encounter these systems. This is done because dj-stripe is not an authentication system.

Example:

Step 1: Add the middleware:

```
MIDDLEWARE_CLASSES = (
    ...
    'djstripe.middleware.SubscriptionPaymentMiddleware',
    ...
    )
```

Step 2: Specify exempt URLS:

```
# sample only - customize to your own needs!
# djstripe pages are automatically exempt!
DJSTRIPE_SUBSCRIPTION_REQUIRED_EXCEPTION_URLS = (
    'home',
    'about',
    "[spam]",  # Anything in the dj-spam namespace
)
```

Using this example any request on this site that isn't on the homepage, about, spam, or djstripe pages is redirected to `djstripe.views.SubscribeFormView`.

Note: The extensive list of rules for this feature can be found at https://github.com/dj-stripe/dj-stripe/blob/master/djstripe/middleware.py.

See also:

- *Settings*

Constraining Class-Based Views

If you want to quickly constrain a single Class-Based View, the `djstripe.decorators.subscription_payment_required` decorator does the following to user requests:

- **authenticated** users are redirected to `djstripe.views.SubscribeFormView` unless they:
  - have a valid subscription –or–
  - are superusers (user.is_superuser==True) –or–
  - are staff members (user.is_staff==True).
- **anonymous** users always raise a `django.core.exceptions.ImproperlyConfigured` exception when they encounter these systems. This is done because dj-stripe is not an authentication system.

Example:

```
# import necessary Django stuff
from django.http import HttpResponse
from django.views.generic import View
from django.contrib.auth.decorators import login_required

# import the wonderful decorator
from djstripe.decorators import subscription_payment_required

# import method_decorator which allows us to use function
# decorators on Class-Based View dispatch function.
from django.utils.decorators import method_decorator


class MyConstrainedView(View):

    def get(self, request, *args, **kwargs):
        return HttpReponse("I like cheese")

    @method_decorator(login_required)
    @method_decorator(subscription_payment_required)
    def dispatch(self, *args, **kwargs):
        return super(MyConstrainedView, self).dispatch(*args, **kwargs)
```

If you are unfamiliar with this technique please read the following documentation here.

Constraining Function-Based Views

If you want to quickly constrain a single Function-Based View, the `djstripe.decorators.subscription_payment_required` decorator does the following to user requests:

- **authenticated** users are redirected to `djstripe.views.SubscribeFormView` unless they:
  - have a valid subscription –or–
  - are superusers (user.is_superuser==True) –or–
  - are staff members (user.is_staff==True).
- **anonymous** users always raise a `django.core.exceptions.ImproperlyConfigured` exception when they encounter these systems. This is done because dj-stripe is not an authentication system.

Example:

```
# import necessary Django stuff
from django.contrib.auth.decorators import login_required
from django.http import HttpResponse

# import the wonderful decorator
from djstripe.decorators import subscription_payment_required

@login_required
@subscription_payment_required
def my_constrained_view(request):
    return HttpReponse("I like cheese")
```

Don't do this!

Described is an anti-pattern. View logic belongs in views.py, not urls.py.

```
# DON'T DO THIS!!!
from django.conf.urls import patterns, url
from django.contrib.auth.decorators import login_required
from djstripe.decorators import subscription_payment_required

from contents import views

urlpatterns = patterns("",

    # Class-Based View anti-pattern
    url(
        r"^content/$",

        # Not using decorators as decorators
        # Harder to see what's going on
        login_required(
            subscription_payment_required(
                views.ContentDetailView.as_view()
            )
        ),
        name="content_detail"
    ),
    # Function-Based View anti-pattern
    url(
        r"^content/$",

        # Example with function view
        login_required(
            subscription_payment_required(
                views.content_list_view
            )
        ),
        name="content_detail"
    ),
)
```

1.2.2 Extending Subscriptions

`Subscription.extend(*delta*)`

Subscriptions can be extended by using the `Subscription.extend` method, which takes a positive `timedelta` as its only property. This method is useful if you want to offer time-cards, gift-cards, or some other external way of subscribing users or extending subscriptions, while keeping the billing handling within Stripe.

> **Warning:** Subscription extensions are achieved by manipulating the `trial_end` of the subscription instance, which means that Stripe will change the status to `trialing`.

1.3 Models

Models hold the bulk of the functionality included in the dj-stripe package. Each model is tied closely to its corresponding object in the stripe dashboard. Fields that are not implemented for each model have a short reason behind the decision in the docstring for each model.

Note: Some model methods documented as classmethods show the base "StripeObject" instead of the model. When using these methods, be sure to replace "StripeObject" with the actual class name.

1.3.1 Charge

class djstripe.models.**Charge**(**args*, ***kwargs*)

To charge a credit or a debit card, you create a charge object. You can retrieve and refund individual charges as well as list all charges. Charges are identified by a unique random ID. (Source: https://stripe.com/docs/api/python#charges)

 # = Mapping the values of this field isn't currently on our roadmap. Please use the stripe dashboard to check the value of this field instead.

Fields not implemented:

- **object** - Unnecessary. Just check the model name.
- **application_fee** - #. Coming soon with stripe connect functionality
- **balance_transaction** - #
- **dispute** - #; Mapped to a `disputed` boolean.
- **order** - #
- **refunds** - #
- **source_transfer** - #

Attention:	Stripe API_VERSION: model fields audited to 2016-06-05 - @jleclanche

Parameters

- **stripe_id** (*StripeIdField*) – Stripe id
- **livemode** (*StripeNullBooleanField*) – Null here indicates that the livemode status is unknown or was previously unrecorded. Otherwise, this field indicates whether this record comes from Stripe test mode or live mode operation.
- **stripe_timestamp** (*StripeDateTimeField*) – The datetime this object was created in stripe.
- **metadata** (*StripeJSONField*) – A set of key/value pairs that you can attach to an object. It can be useful for storing additional information about an object in a structured format.
- **description** (*StripeTextField*) – A description of this object.
- **amount** (*StripeCurrencyField*) – Amount charged.

- **amount_refunded** (*StripeCurrencyField*) – Amount refunded (can be less than the amount attribute on the charge if a partial refund was issued).
- **captured** (*StripeBooleanField*) – If the charge was created without capturing, this boolean represents whether or not it is still uncaptured or has since been captured.
- **currency** (*StripeCharField*) – Three-letter ISO currency code representing the currency in which the charge was made.
- **customer** (ForeignKey to *Customer*) – The customer associated with this charge.
- **account** (ForeignKey to Account) – The account the charge was made on behalf of. Null here indicates that this value was never set.
- **failure_code** (*StripeCharField*) – Error code explaining reason for charge failure if available.
- **failure_message** (*StripeTextField*) – Message to user further explaining reason for charge failure if available.
- **fraud_details** (*StripeJSONField*) – Hash with information on fraud assessments for the charge.
- **invoice** (ForeignKey to *Invoice*) – The invoice this charge is for if one exists.
- **outcome** (*StripeJSONField*) – Details about whether or not the payment was accepted, and why.
- **paid** (*StripeBooleanField*) – True if the charge succeeded, or was successfully authorized for later capture, False otherwise.
- **receipt_email** (*StripeCharField*) – The email address that the receipt for this charge was sent to.
- **receipt_number** (*StripeCharField*) – The transaction number that appears on email receipts sent for this charge.
- **refunded** (*StripeBooleanField*) – Whether or not the charge has been fully refunded. If the charge is only partially refunded, this attribute will still be false.
- **shipping** (*StripeJSONField*) – Shipping information for the charge
- **source** (ForeignKey to StripeSource) – The source used for this charge.
- **statement_descriptor** (*StripeCharField*) – An arbitrary string to be displayed on your customer's credit card statement. The statement description may not include <>"' characters, and will appear on your customer's statement in capital letters. Non-ASCII characters are automatically stripped. While most banks display this information consistently, some may display it incorrectly or not at all.
- **status** (*StripeCharField*) – The status of the payment.
- **transfer** (ForeignKey to *Transfer*) – The transfer to the destination account (only applicable if the charge was created using the destination parameter).
- **fee** (*StripeCurrencyField*) – Fee
- **fee_details** (*StripeJSONField*) – Fee details
- **source_type** (*StripeCharField*) – The payment source type. If the payment source is supported by dj-stripe, a corresponding model is attached to this Charge via a foreign key matching this field.
- **source_stripe_id** (*StripeIdField*) – The payment source id.

- **disputed** (*StripeBooleanField*) – Whether or not this charge is disputed.
- **fraudulent** (*StripeBooleanField*) – Whether or not this charge was marked as fraudulent.
- **receipt_sent** (*BooleanField*) – Whether or not a receipt was sent for this charge.

api_retrieve(*api_key=None*)
Call the stripe API's retrieve operation for this model.

Parameters **api_key** (*string*) – The api key to use for this request. Defaults to settings.STRIPE_SECRET_KEY.

capture()
Capture the payment of an existing, uncaptured, charge. This is the second half of the two-step payment flow, where first you created a charge with the capture option set to False.

See https://stripe.com/docs/api#capture_charge

refund(*amount=None*, *reason=None*)
Initiate a refund. If amount is not provided, then this will be a full refund.

Parameters

- **amount** – A positive decimal amount representing how much of this charge to refund. Can only refund up to the unrefunded amount remaining of the charge.
- **reason** – String indicating the reason for the refund. If set, possible values are `duplicate`, `fraudulent`, and `requested_by_customer`. Specifying `fraudulent` as the reason when you believe the charge to be fraudulent will help Stripe improve their fraud detection algorithms.

Trye amount Decimal

Returns Stripe charge object

Return type dict

str_parts()

1.3.2 Customer

class djstripe.models.**Customer**(**args*, ***kwargs*)
Customer objects allow you to perform recurring charges and track multiple charges that are associated with the same customer. (Source: https://stripe.com/docs/api/python#customers)

 # = Mapping the values of this field isn't currently on our roadmap. Please use the stripe dashboard to check the value of this field instead.

Fields not implemented:

- **object** - Unnecessary. Just check the model name.
- **discount** - #

> **Attention:** Stripe API_VERSION: model fields and methods audited to 2017-06-05 - @jleclanche

Parameters

- **stripe_id** (*StripeIdField*) – Stripe id

- **livemode** (*StripeNullBooleanField*) – Null here indicates that the livemode status is unknown or was previously unrecorded. Otherwise, this field indicates whether this record comes from Stripe test mode or live mode operation.
- **stripe_timestamp** (*StripeDateTimeField*) – The datetime this object was created in stripe.
- **metadata** (*StripeJSONField*) – A set of key/value pairs that you can attach to an object. It can be useful for storing additional information about an object in a structured format.
- **description** (*StripeTextField*) – A description of this object.
- **account_balance** (*StripeIntegerField*) – Current balance, if any, being stored on the customer's account. If negative, the customer has credit to apply to the next invoice. If positive, the customer has an amount owed that will be added to thenext invoice. The balance does not refer to any unpaid invoices; it solely takes into account amounts that have yet to be successfullyapplied to any invoice. This balance is only taken into account for recurring billing purposes (i.e., subscriptions, invoices, invoice items).
- **business_vat_id** (*StripeCharField*) – The customer's VAT identification number.
- **currency** (*StripeCharField*) – The currency the customer can be charged in for recurring billing purposes (subscriptions, invoices, invoice items).
- **default_source** (ForeignKey to `StripeSource`) – Default source
- **delinquent** (*StripeBooleanField*) – Whether or not the latest charge for the customer's latest invoice has failed.
- **coupon** (ForeignKey to `Coupon`) – Coupon
- **coupon_start** (*StripeDateTimeField*) – If a coupon is present, the date at which it was applied.
- **coupon_end** (*StripeDateTimeField*) – If a coupon is present and has a limited duration, the date that the discount will end.
- **email** (*StripeTextField*) – Email
- **shipping** (*StripeJSONField*) – Shipping information associated with the customer.
- **subscriber** (ForeignKey to `User`) – Subscriber
- **date_purged** (*DateTimeField*) – Date purged

api_retrieve(*api_key=None*)

Call the stripe API's retrieve operation for this model.

Parameters **api_key** (*string*) – The api key to use for this request. Defaults to settings.STRIPE_SECRET_KEY.

classmethod get_or_create(*subscriber*, *livemode=False*)

Get or create a dj-stripe customer.

Parameters

- **subscriber** (*User*) – The subscriber model instance for which to get or create a customer.
- **livemode** (*bool*) – Whether to get the subscriber in live or test mode.

purge()

has_active_subscription(*plan=None*)
Checks to see if this customer has an active subscription to the given plan.

Parameters **plan** (*Plan or string (plan ID)*) – The plan for which to check for an active subscription. If plan is None and there exists only one active subscription, this method will check if that subscription is valid. Calling this method with no plan and multiple valid subscriptions for this customer will throw an exception.

Returns True if there exists an active subscription, False otherwise.

Throws TypeError if `plan` is None and more than one active subscription exists for this customer.

has_any_active_subscription()
Checks to see if this customer has an active subscription to any plan.

Returns True if there exists an active subscription, False otherwise.

Throws TypeError if `plan` is None and more than one active subscription exists for this customer.

subscription
Shortcut to get this customer's subscription.

Returns None if the customer has no subscriptions, the subscription if the customer has a subscription.

Raises **MultipleSubscriptionException** – Raised if the customer has multiple subscriptions. In this case, use `Customer.subscriptions` instead.

subscribe(*plan*, *charge_immediately=True*, *application_fee_percent=None*, *coupon=None*, *quantity=None*, *metadata=None*, *tax_percent=None*, *trial_end=None*)
Subscribes this customer to a plan.

Parameters not implemented:

- **source** - Subscriptions use the customer's default source. Including the source parameter creates a new source for this customer and overrides the default source. This functionality is not desired; add a source to the customer before attempting to add a subscription.

Parameters

- **plan** (*Plan or string (plan ID)*) – The plan to which to subscribe the customer.
- **application_fee_percent** (*Decimal. Precision is 2; anything more will be ignored. A positive decimal between 1 and 100.*) – This represents the percentage of the subscription invoice subtotal that will be transferred to the application owner's Stripe account. The request must be made with an OAuth key in order to set an application fee percentage.
- **coupon** (*string*) – The code of the coupon to apply to this subscription. A coupon applied to a subscription will only affect invoices created for that particular subscription.
- **quantity** (*integer*) – The quantity applied to this subscription. Default is 1.
- **metadata** (*dict*) – A set of key/value pairs useful for storing additional information.
- **tax_percent** (*Decimal. Precision is 2; anything more will be ignored. A positive decimal between 1 and 100.*) – This represents the percentage of the subscription invoice subtotal that will be calculated and added as tax to the final amount each billing period.

- **trial_end** (*datetime*) – The end datetime of the trial period the customer will get before being charged for the first time. If set, this will override the default trial period of the plan the customer is being subscribed to. The special value `now` can be provided to end the customer's trial immediately.
- **charge_immediately** (*boolean*) – Whether or not to charge for the subscription upon creation. If False, an invoice will be created at the end of this period.

can_charge()
Determines if this customer is able to be charged.

charge(*amount*, *currency=None*, *application_fee=None*, *capture=None*, *description=None*, *destination=None*, *metadata=None*, *shipping=None*, *source=None*, *statement_descriptor=None*)
Creates a charge for this customer.

Parameters not implemented:

- **receipt_email** - Since this is a charge on a customer, the customer's email address is used.

Parameters

- **amount** (*Decimal. Precision is 2; anything more will be ignored.*) – The amount to charge.
- **currency** (*string*) – 3-letter ISO code for currency
- **application_fee** (*Decimal. Precision is 2; anything more will be ignored.*) – A fee that will be applied to the charge and transfered to the platform owner's account.
- **capture** (*bool*) – Whether or not to immediately capture the charge. When false, the charge issues an authorization (or pre-authorization), and will need to be captured later. Uncaptured charges expire in 7 days. Default is True
- **description** (*string*) – An arbitrary string.
- **destination** (*Account*) – An account to make the charge on behalf of.
- **metadata** (*dict*) – A set of key/value pairs useful for storing additional information.
- **shipping** (*dict*) – Shipping information for the charge.
- **source** (*string, Source*) – The source to use for this charge. Must be a source attributed to this customer. If None, the customer's default source is used. Can be either the id of the source or the source object itself.
- **statement_descriptor** (*string*) – An arbitrary string to be displayed on the customer's credit card statement.

add_invoice_item(*amount*, *currency*, *description=None*, *discountable=None*, *invoice=None*, *metadata=None*, *subscription=None*)
Adds an arbitrary charge or credit to the customer's upcoming invoice. Different than creating a charge. Charges are separate bills that get processed immediately. Invoice items are appended to the customer's next invoice. This is extremely useful when adding surcharges to subscriptions.

Parameters

- **amount** (*Decimal. Precision is 2; anything more will be ignored.*) – The amount to charge.
- **currency** (*string*) – 3-letter ISO code for currency
- **description** (*string*) – An arbitrary string.

- **discountable** (*boolean*) – Controls whether discounts apply to this invoice item. Defaults to False for prorations or negative invoice items, and True for all other invoice items.
- **invoice** (Invoice *or string (invoice ID)*) – An existing invoice to add this invoice item to. When left blank, the invoice item will be added to the next upcoming scheduled invoice. Use this when adding invoice items in response to an `invoice.created` webhook. You cannot add an invoice item to an invoice that has already been paid, attempted or closed.
- **metadata** (*dict*) – A set of key/value pairs useful for storing additional information.
- **subscription** (Subscription *or string (subscription ID)*) – A subscription to add this invoice item to. When left blank, the invoice item will be be added to the next upcoming scheduled invoice. When set, scheduled invoices for subscriptions other than the specified subscription will ignore the invoice item. Use this when you want to express that an invoice item has been accrued within the context of a particular subscription.

send_invoice()
Pay and send the customer's latest invoice.

Returns True if an invoice was able to be created and paid, False otherwise (typically if there was nothing to invoice).

retry_unpaid_invoices()
Attempt to retry collecting payment on the customer's unpaid invoices.

has_valid_source()
Check whether the customer has a valid payment source.

add_card(*source*, *set_default=True*)
Adds a card to this customer's account.

Parameters

- **source** (*string, dict*) – Either a token, like the ones returned by our Stripe.js, or a dictionary containing a user's credit card details. Stripe will automatically validate the card.
- **set_default** (*boolean*) – Whether or not to set the source as the customer's default source

upcoming_invoice(***kwargs*)
Gets the upcoming preview invoice (singular) for this customer.

See *Invoice.upcoming()*.

The `customer` argument to the `upcoming()` call is automatically set by this method.

str_parts()
Extend this to add information to the string representation of the object

Return type list of str

1.3.3 Event

class djstripe.models.**Event**(**args*, ***kwargs*)
Events are POSTed to our webhook url. They provide information about a Stripe event that just happened. Events are processed in detail by their respective models (charge events by the Charge model, etc).

Events are initially **UNTRUSTED**, as it is possible for any web entity to post any data to our webhook url. Data posted may be valid Stripe information, garbage, or even malicious. The 'valid' flag in this model monitors this.

API VERSIONING

This is a tricky matter when it comes to webhooks. See the discussion here.

In this discussion, it is noted that Webhooks are produced in one API version, which will usually be different from the version supported by Stripe plugins (such as djstripe). The solution, described there, is:

1. validate the receipt of a webhook event by doing an event get using the API version of the received hook event.
2. retrieve the referenced object (e.g. the Charge, the Customer, etc) using the plugin's supported API version.

3) process that event using the retrieved object which will, only now, be in a format that you are certain to understand

 # = Mapping the values of this field isn't currently on our roadmap. Please use the stripe dashboard to check the value of this field instead.

Fields not implemented:

- **object** - Unnecessary. Just check the model name.
- **pending_webhooks** - Unnecessary. Use the dashboard.

Attention:	Stripe API_VERSION: model fields and methods audited to 2016-03-07 - @kavdev

Parameters

- **stripe_id** (*StripeIdField*) – Stripe id
- **livemode** (*StripeNullBooleanField*) – Null here indicates that the livemode status is unknown or was previously unrecorded. Otherwise, this field indicates whether this record comes from Stripe test mode or live mode operation.
- **stripe_timestamp** (*StripeDateTimeField*) – The datetime this object was created in stripe.
- **metadata** (*StripeJSONField*) – A set of key/value pairs that you can attach to an object. It can be useful for storing additional information about an object in a structured format.
- **description** (*StripeTextField*) – A description of this object.
- **received_api_version** (*StripeCharField*) – the API version at which the event data was rendered. Blank for old entries only, all new entries will have this value
- **webhook_message** (*StripeJSONField*) – data received at webhook. data should be considered to be garbage until validity check is run and valid flag is set
- **request_id** (*StripeCharField*) – Information about the request that triggered this event, for traceability purposes. If empty string then this is an old entry without that data. If Null then this is not an old entry, but a Stripe 'automated' event with no associated request.
- **idempotency_key** (*StripeTextField*) – Idempotency key
- **type** (*StripeCharField*) – Stripe's event description code
- **customer** (ForeignKey to *Customer*) – In the event that there is a related customer, this will point to that Customer record

- **valid** (*NullBooleanField*) – Tri-state bool. Null == validity not yet confirmed. Otherwise, this field indicates that this event was checked via stripe api and found to be either authentic (valid=True) or in-authentic (possibly malicious)
- **processed** (*BooleanField*) – If validity is performed, webhook event processor(s) may run to take further action on the event. Once these have run, this is set to True.

api_retrieve(*api_key=None*)

message
The event's data if the event is valid, None otherwise.

validate()
The original contents of the Event message comes from a POST to the webhook endpoint. This data must be confirmed by re-fetching it and comparing the fetched data with the original data. That's what this function does.

This function makes an API call to Stripe to re-download the Event data. It then marks this record's valid flag to True or False.

process(*force=False*, *raise_exception=False*)
Invokes any webhook handlers that have been registered for this event based on event type or event sub-type.

See event handlers registered in the djstripe.event_handlers module (or handlers registered in djstripe plugins or contrib packages).

Parameters **force** – If True, force the event to be processed by webhook

handlers, even if the event has already been processed previously. :type force: bool :param raise_exception: If True, any Stripe errors raised during processing will be raised to the caller after logging the exception. :type raise_exception: bool :returns: True if the webhook was processed successfully or was previously processed successfully. :rtype: bool

str_parts()

1.3.4 Transfer

class djstripe.models.**Transfer**(**args*, ***kwargs*)
When Stripe sends you money or you initiate a transfer to a bank account, debit card, or connected Stripe account, a transfer object will be created. (Source: https://stripe.com/docs/api/python#transfers)

 # = Mapping the values of this field isn't currently on our roadmap. Please use the stripe dashboard to check the value of this field instead.

Fields not implemented:

- **object** - Unnecessary. Just check the model name.
- **application_fee** - #
- **balance_transaction** - #
- **reversals** - #

Attention:	Stripe API_VERSION: model fields and methods audited to 2016-03-07 - @kavdev

Parameters

- **stripe_id** (*StripeIdField*) – Stripe id

- **livemode** (*StripeNullBooleanField*) – Null here indicates that the livemode status is unknown or was previously unrecorded. Otherwise, this field indicates whether this record comes from Stripe test mode or live mode operation.
- **stripe_timestamp** (*StripeDateTimeField*) – The datetime this object was created in stripe.
- **metadata** (*StripeJSONField*) – A set of key/value pairs that you can attach to an object. It can be useful for storing additional information about an object in a structured format.
- **description** (*StripeTextField*) – A description of this object.
- **amount** (*StripeCurrencyField*) – The amount transferred
- **amount_reversed** (*StripeCurrencyField*) – The amount reversed (can be less than the amount attribute on the transfer if a partial reversal was issued).
- **currency** (*StripeCharField*) – Three-letter ISO currency code.
- **date** (*StripeDateTimeField*) – Date the transfer is scheduled to arrive in the bank. This doesn't factor in delays like weekends or bank holidays.
- **destination** (*StripeIdField*) – ID of the bank account, card, or Stripe account the transfer was sent to.
- **destination_payment** (*StripeIdField*) – If the destination is a Stripe account, this will be the ID of the payment that the destination account received for the transfer.
- **destination_type** (*StripeCharField*) – The type of the transfer destination.
- **failure_code** (*StripeCharField*) – Error code explaining reason for transfer failure if available. See https://stripe.com/docs/api/python#transfer_failures.
- **failure_message** (*StripeTextField*) – Message to user further explaining reason for transfer failure if available.
- **reversed** (*StripeBooleanField*) – Whether or not the transfer has been fully reversed. If the transfer is only partially reversed, this attribute will still be false.
- **source_transaction** (*StripeIdField*) – ID of the charge (or other transaction) that was used to fund the transfer. If null, the transfer was funded from the available balance.
- **source_type** (*StripeCharField*) – The source balance from which this transfer came.
- **statement_descriptor** (*StripeCharField*) – An arbitrary string to be displayed on your customer's credit card statement. The statement description may not include <>"' characters, and will appear on your customer's statement in capital letters. Non-ASCII characters are automatically stripped. While most banks display this information consistently, some may display it incorrectly or not at all.
- **status** (*StripeCharField*) – The current status of the transfer. A transfer will be pending until it is submitted to the bank, at which point it becomes in_transit. It will then change to paid if the transaction goes through. If it does not go through successfully, its status will change to failed or canceled.
- **fee** (*StripeCurrencyField*) – Fee
- **fee_details** (*StripeJSONField*) – Fee details

api_retrieve(*api_key=None*)

Call the stripe API's retrieve operation for this model.

Parameters **api_key** (*string*) – The api key to use for this request. Defaults to settings.STRIPE_SECRET_KEY.

DESTINATION_TYPES = ['card', 'bank_account', 'stripe_account']

str_parts()

1.3.5 Card

class djstripe.models.**Card**(**args*, ***kwargs*)

You can store multiple cards on a customer in order to charge the customer later. (Source: https://stripe.com/docs/api/python#cards)

 # = Mapping the values of this field isn't currently on our roadmap. Please use the stripe dashboard to check the value of this field instead.

Fields not implemented:

- **object** - Unnecessary. Just check the model name.
- **recipient** - On Stripe's deprecation path.
- **account** - #
- **currency** - #
- **default_for_currency** - #

Attention:	Stripe API_VERSION: model fields and methods audited to 2016-03-07 - @kavdev

Parameters

- **stripe_id** (*StripeIdField*) – Stripe id
- **livemode** (*StripeNullBooleanField*) – Null here indicates that the livemode status is unknown or was previously unrecorded. Otherwise, this field indicates whether this record comes from Stripe test mode or live mode operation.
- **stripe_timestamp** (*StripeDateTimeField*) – The datetime this object was created in stripe.
- **metadata** (*StripeJSONField*) – A set of key/value pairs that you can attach to an object. It can be useful for storing additional information about an object in a structured format.
- **description** (*StripeTextField*) – A description of this object.
- **customer** (ForeignKey to *Customer*) – Customer
- **stripesource_ptr** (OneToOneField to StripeSource) – Stripesource ptr
- **address_city** (*StripeTextField*) – Billing address city.
- **address_country** (*StripeTextField*) – Billing address country.
- **address_line1** (*StripeTextField*) – Billing address (Line 1).
- **address_line1_check** (*StripeCharField*) – If address_line1 was provided, results of the check.
- **address_line2** (*StripeTextField*) – Billing address (Line 2).
- **address_state** (*StripeTextField*) – Billing address state.

- **address_zip** (*StripeTextField*) – Billing address zip code.
- **address_zip_check** (*StripeCharField*) – If `address_zip` was provided, results of the check.
- **brand** (*StripeCharField*) – Card brand.
- **country** (*StripeCharField*) – Two-letter ISO code representing the country of the card.
- **cvc_check** (*StripeCharField*) – If a CVC was provided, results of the check.
- **dynamic_last4** (*StripeCharField*) – (For tokenized numbers only.) The last four digits of the device account number.
- **exp_month** (*StripeIntegerField*) – Card expiration month.
- **exp_year** (*StripeIntegerField*) – Card expiration year.
- **fingerprint** (*StripeTextField*) – Uniquely identifies this particular card number.
- **funding** (*StripeCharField*) – Card funding type.
- **last4** (*StripeCharField*) – Last four digits of Card number.
- **name** (*StripeTextField*) – Cardholder name.
- **tokenization_method** (*StripeCharField*) – If the card number is tokenized, this is the method that was used.

api_retrieve(*api_key=None*)

remove()
Removes a card from this customer's account.

str_parts()

1.3.6 Invoice

class djstripe.models.**Invoice**(**args*, ***kwargs*)
Invoices are statements of what a customer owes for a particular billing period, including subscriptions, invoice items, and any automatic proration adjustments if necessary.

Once an invoice is created, payment is automatically attempted. Note that the payment, while automatic, does not happen exactly at the time of invoice creation. If you have configured webhooks, the invoice will wait until one hour after the last webhook is successfully sent (or the last webhook times out after failing).

Any customer credit on the account is applied before determining how much is due for that invoice (the amount that will be actually charged). If the amount due for the invoice is less than 50 cents (the minimum for a charge), we add the amount to the customer's running account balance to be added to the next invoice. If this amount is negative, it will act as a credit to offset the next invoice. Note that the customer account balance does not include unpaid invoices; it only includes balances that need to be taken into account when calculating the amount due for the next invoice. (Source: https://stripe.com/docs/api/python#invoices)

 # = Mapping the values of this field isn't currently on our roadmap. Please use the stripe dashboard to check the value of this field instead.

Fields not implemented:

- **object** - Unnecessary. Just check the model name.
- **discount** - #
- **lines** - Unnecessary. Check Subscription and InvoiceItems directly.

- **webhooks_delivered_at** - #

Attention: Stripe API_VERSION: model fields audited to 2017-06-05 - @jleclanche

Parameters

- **stripe_id** (*StripeIdField*) – Stripe id
- **livemode** (*StripeNullBooleanField*) – Null here indicates that the livemode status is unknown or was previously unrecorded. Otherwise, this field indicates whether this record comes from Stripe test mode or live mode operation.
- **stripe_timestamp** (*StripeDateTimeField*) – The datetime this object was created in stripe.
- **metadata** (*StripeJSONField*) – A set of key/value pairs that you can attach to an object. It can be useful for storing additional information about an object in a structured format.
- **description** (*StripeTextField*) – A description of this object.
- **amount_due** (*StripeCurrencyField*) – Final amount due at this time for this invoice. If the invoice's total is smaller than the minimum charge amount, for example, or if there is account credit that can be applied to the invoice, the amount_due may be 0. If there is a positive starting_balance for the invoice (the customer owes money), the amount_due will also take that into account. The charge that gets generated for the invoice will be for the amount specified in amount_due.
- **application_fee** (*StripeCurrencyField*) – The fee in cents that will be applied to the invoice and transferred to the application owner's Stripe account when the invoice is paid.
- **attempt_count** (*StripeIntegerField*) – Number of payment attempts made for this invoice, from the perspective of the payment retry schedule. Any payment attempt counts as the first attempt, and subsequently only automatic retries increment the attempt count. In other words, manual payment attempts after the first attempt do not affect the retry schedule.
- **attempted** (*StripeBooleanField*) – Whether or not an attempt has been made to pay the invoice. An invoice is not attempted until 1 hour after the `invoice.created` webhook, for example, so you might not want to display that invoice as unpaid to your users.
- **charge** (OneToOneField to *Charge*) – The latest charge generated for this invoice, if any.
- **closed** (*StripeBooleanField*) – Whether or not the invoice is still trying to collect payment. An invoice is closed if it's either paid or it has been marked closed. A closed invoice will no longer attempt to collect payment.
- **currency** (*StripeCharField*) – Three-letter ISO currency code.
- **customer** (ForeignKey to *Customer*) – The customer associated with this invoice.
- **date** (*StripeDateTimeField*) – The date on the invoice.
- **ending_balance** (*StripeIntegerField*) – Ending customer balance after attempting to pay invoice. If the invoice has not been attempted yet, this will be null.

- **forgiven** (*StripeBooleanField*) – Whether or not the invoice has been forgiven. Forgiving an invoice instructs us to update the subscription status as if the invoice were successfully paid. Once an invoice has been forgiven, it cannot be unforgiven or reopened.
- **next_payment_attempt** (*StripeDateTimeField*) – The time at which payment will next be attempted.
- **paid** (*StripeBooleanField*) – The time at which payment will next be attempted.
- **period_end** (*StripeDateTimeField*) – End of the usage period during which invoice items were added to this invoice.
- **period_start** (*StripeDateTimeField*) – Start of the usage period during which invoice items were added to this invoice.
- **starting_balance** (*StripeIntegerField*) – Starting customer balance before attempting to pay invoice. If the invoice has not been attempted yet, this will be the current customer balance.
- **statement_descriptor** (*StripeCharField*) – An arbitrary string to be displayed on your customer's credit card statement. The statement description may not include <>"' characters, and will appear on your customer's statement in capital letters. Non-ASCII characters are automatically stripped. While most banks display this information consistently, some may display it incorrectly or not at all.
- **subscription** (ForeignKey to *Subscription*) – The subscription that this invoice was prepared for, if any.
- **subscription_proration_date** (*StripeDateTimeField*) – Only set for upcoming invoices that preview prorations. The time used to calculate prorations.
- **subtotal** (*StripeCurrencyField*) – Only set for upcoming invoices that preview prorations. The time used to calculate prorations.
- **tax** (*StripeCurrencyField*) – The amount of tax included in the total, calculated from `tax_percent` and the subtotal. If no `tax_percent` is defined, this value will be null.
- **tax_percent** (*StripePercentField*) – This percentage of the subtotal has been added to the total amount of the invoice, including invoice line items and discounts. This field is inherited from the subscription's `tax_percent` field, but can be changed before the invoice is paid. This field defaults to null.
- **total** (*StripeCurrencyField*) – Total after discount.

api_retrieve(*api_key=None*)

Call the stripe API's retrieve operation for this model.

Parameters **api_key** (*string*) – The api key to use for this request. Defaults to settings.STRIPE_SECRET_KEY.

STATUS_PAID = 'Paid'

STATUS_FORGIVEN = 'Forgiven'

STATUS_CLOSED = 'Closed'

STATUS_OPEN = 'Open'

status

Attempts to label this invoice with a status. Note that an invoice can be more than one of the choices. We just set a priority on which status appears.

plan
Gets the associated plan for this invoice.

In order to provide a consistent view of invoices, the plan object should be taken from the first invoice item that has one, rather than using the plan associated with the subscription.

Subscriptions (and their associated plan) are updated by the customer and represent what is current, but invoice items are immutable within the invoice and stay static/unchanged.

In other words, a plan retrieved from an invoice item will represent the plan as it was at the time an invoice was issued. The plan retrieved from the subscription will be the currently active plan.

Returns The associated plan for the invoice.

Return type djstripe.Plan

retry()
Retry payment on this invoice if it isn't paid, closed, or forgiven.

classmethod **upcoming**(*api_key=''*, *customer=None*, *coupon=None*, *subscription=None*, *subscription_plan=None*, *subscription_prorate=None*, *subscription_proration_date=None*, *subscription_quantity=None*, *subscription_trial_end=None*, ***kwargs*)
Gets the upcoming preview invoice (singular) for a customer.

At any time, you can preview the upcoming invoice for a customer. This will show you all the charges that are pending, including subscription renewal charges, invoice item charges, etc. It will also show you any discount that is applicable to the customer. (Source: https://stripe.com/docs/api#upcoming_invoice)

Important: Note that when you are viewing an upcoming invoice, you are simply viewing a preview.

Parameters

- **customer** (*Customer or string (customer ID)*) – The identifier of the customer whose upcoming invoice you'd like to retrieve.
- **coupon** (*str*) – The code of the coupon to apply.
- **subscription** (*Subscription or string (subscription ID)*) – The identifier of the subscription to retrieve an invoice for.
- **subscription_plan** (*Plan or string (plan ID)*) – If set, the invoice returned will preview updating the subscription given to this plan, or creating a new subscription to this plan if no subscription is given.
- **subscription_prorate** (*bool*) – If previewing an update to a subscription, this decides whether the preview will show the result of applying prorations or not.
- **subscription_proration_date** (*datetime*) – If previewing an update to a subscription, and doing proration, subscription_proration_date forces the proration to be calculated as though the update was done at the specified time.
- **subscription_quantity** (*int*) – If provided, the invoice returned will preview updating or creating a subscription with that quantity.
- **subscription_trial_end** (*datetime*) – If provided, the invoice returned will preview updating or creating a subscription with that trial end.

Returns The upcoming preview invoice.

Return type UpcomingInvoice

str_parts()

1.3.7 InvoiceItem

class djstripe.models.**InvoiceItem**(**args*, ***kwargs*)

Sometimes you want to add a charge or credit to a customer but only actually charge the customer's card at the end of a regular billing cycle. This is useful for combining several charges to minimize per-transaction fees or having Stripe tabulate your usage-based billing totals. (Source: https://stripe.com/docs/api/python#invoiceitems)

 # = Mapping the values of this field isn't currently on our roadmap. Please use the stripe dashboard to check the value of this field instead.

Fields not implemented:

- **object** - Unnecessary. Just check the model name.

Attention:	Stripe API_VERSION: model fields audited to 2017-06-05 - @jleclanche

Parameters

- **stripe_id** (*StripeIdField*) – Stripe id
- **livemode** (*StripeNullBooleanField*) – Null here indicates that the livemode status is unknown or was previously unrecorded. Otherwise, this field indicates whether this record comes from Stripe test mode or live mode operation.
- **stripe_timestamp** (*StripeDateTimeField*) – The datetime this object was created in stripe.
- **metadata** (*StripeJSONField*) – A set of key/value pairs that you can attach to an object. It can be useful for storing additional information about an object in a structured format.
- **description** (*StripeTextField*) – A description of this object.
- **amount** (*StripeCurrencyField*) – Amount invoiced.
- **currency** (*StripeCharField*) – Three-letter ISO currency code.
- **customer** (ForeignKey to *Customer*) – The customer associated with this invoiceitem.
- **date** (*StripeDateTimeField*) – The date on the invoiceitem.
- **discountable** (*StripeBooleanField*) – If True, discounts will apply to this invoice item. Always False for prorations.
- **invoice** (ForeignKey to *Invoice*) – The invoice to which this invoiceitem is attached.
- **period_end** (*StripeDateTimeField*) – Might be the date when this invoiceitem's invoice was sent.
- **period_start** (*StripeDateTimeField*) – Might be the date when this invoiceitem was added to the invoice
- **plan** (ForeignKey to *Plan*) – If the invoice item is a proration, the plan of the subscription for which the proration was computed.
- **proration** (*StripeBooleanField*) – Whether or not the invoice item was created automatically as a proration adjustment when the customer switched plans.

- **quantity** (*StripeIntegerField*) – If the invoice item is a proration, the quantity of the subscription for which the proration was computed.
- **subscription** (ForeignKey to *Subscription*) – The subscription that this invoice item has been created for, if any.

api_retrieve(*api_key=None*)
Call the stripe API's retrieve operation for this model.

Parameters **api_key** (*string*) – The api key to use for this request. Defaults to settings.STRIPE_SECRET_KEY.

str_parts()

1.3.8 Plan

class djstripe.models.**Plan**(**args*, ***kwargs*)
A subscription plan contains the pricing information for different products and feature levels on your site. (Source: https://stripe.com/docs/api/python#plans)

= Mapping the values of this field isn't currently on our roadmap. Please use the stripe dashboard to check the value of this field instead.

Fields not implemented:

- **object** - Unnecessary. Just check the model name.

Attention:	Stripe API_VERSION: model fields and methods audited to 2016-03-07 - @kavdev

Parameters

- **stripe_id** (*StripeIdField*) – Stripe id
- **livemode** (*StripeNullBooleanField*) – Null here indicates that the livemode status is unknown or was previously unrecorded. Otherwise, this field indicates whether this record comes from Stripe test mode or live mode operation.
- **stripe_timestamp** (*StripeDateTimeField*) – The datetime this object was created in stripe.
- **metadata** (*StripeJSONField*) – A set of key/value pairs that you can attach to an object. It can be useful for storing additional information about an object in a structured format.
- **description** (*StripeTextField*) – A description of this object.
- **amount** (*StripeCurrencyField*) – Amount to be charged on the interval specified.
- **currency** (*StripeCharField*) – Three-letter ISO currency code
- **interval** (*StripeCharField*) – The frequency with which a subscription should be billed.
- **interval_count** (*StripeIntegerField*) – The number of intervals (specified in the interval property) between each subscription billing.
- **name** (*StripeTextField*) – Name of the plan, to be displayed on invoices and in the web interface.

- **statement_descriptor** (*StripeCharField*) – An arbitrary string to be displayed on your customer's credit card statement. The statement description may not include <>"' characters, and will appear on your customer's statement in capital letters. Non-ASCII characters are automatically stripped. While most banks display this information consistently, some may display it incorrectly or not at all.
- **trial_period_days** (*StripeIntegerField*) – Number of trial period days granted when subscribing a customer to this plan. Null if the plan has no trial period.

api_retrieve(*api_key=None*)
Call the stripe API's retrieve operation for this model.

Parameters **api_key** (*string*) – The api key to use for this request. Defaults to settings.STRIPE_SECRET_KEY.

classmethod get_or_create(***kwargs*)
Get or create a Plan.

str_parts()
Extend this to add information to the string representation of the object

Return type list of str

1.3.9 Subscription

class djstripe.models.**Subscription**(**args*, ***kwargs*)
Subscriptions allow you to charge a customer's card on a recurring basis. A subscription ties a customer to a particular plan you've created.

A subscription still in its trial period is `trialing` and moves to `active` when the trial period is over. When payment to renew the subscription fails, the subscription becomes `past_due`. After Stripe has exhausted all payment retry attempts, the subscription ends up with a status of either `canceled` or `unpaid` depending on your retry settings. Note that when a subscription has a status of `unpaid`, no subsequent invoices will be attempted (invoices will be created, but then immediately automatically closed. Additionally, updating customer card details will not lead to Stripe retrying the latest invoice.). After receiving updated card details from a customer, you may choose to reopen and pay their closed invoices. (Source: https://stripe.com/docs/api/python#subscriptions)

 # = Mapping the values of this field isn't currently on our roadmap. Please use the stripe dashboard to check the value of this field instead.

Fields not implemented:

- **object** - Unnecessary. Just check the model name.
- **discount** - #

Attention:	Stripe API_VERSION: model fields and methods audited to 2016-03-07 - @kavdev

Parameters

- **stripe_id** (*StripeIdField*) – Stripe id
- **livemode** (*StripeNullBooleanField*) – Null here indicates that the livemode status is unknown or was previously unrecorded. Otherwise, this field indicates whether this record comes from Stripe test mode or live mode operation.

- **stripe_timestamp** (*StripeDateTimeField*) – The datetime this object was created in stripe.
- **metadata** (*StripeJSONField*) – A set of key/value pairs that you can attach to an object. It can be useful for storing additional information about an object in a structured format.
- **description** (*StripeTextField*) – A description of this object.
- **application_fee_percent** (*StripePercentField*) – A positive decimal that represents the fee percentage of the subscription invoice amount that will be transferred to the application owner's Stripe account each billing period.
- **cancel_at_period_end** (*StripeBooleanField*) – If the subscription has been canceled with the `at_period_end` flag set to true, `cancel_at_period_end` on the subscription will be true. You can use this attribute to determine whether a subscription that has a status of active is scheduled to be canceled at the end of the current period.
- **canceled_at** (*StripeDateTimeField*) – If the subscription has been canceled, the date of that cancellation. If the subscription was canceled with `cancel_at_period_end`, canceled_at will still reflect the date of the initial cancellation request, not the end of the subscription period when the subscription is automatically moved to a canceled state.
- **current_period_end** (*StripeDateTimeField*) – End of the current period for which the subscription has been invoiced. At the end of this period, a new invoice will be created.
- **current_period_start** (*StripeDateTimeField*) – Start of the current period for which the subscription has been invoiced.
- **customer** (ForeignKey to *Customer*) – The customer associated with this subscription.
- **ended_at** (*StripeDateTimeField*) – If the subscription has ended (either because it was canceled or because the customer was switched to a subscription to a new plan), the date the subscription ended.
- **plan** (ForeignKey to *Plan*) – The plan associated with this subscription.
- **quantity** (*StripeIntegerField*) – The quantity applied to this subscription.
- **start** (*StripeDateTimeField*) – Date the subscription started.
- **status** (*StripeCharField*) – The status of this subscription.
- **tax_percent** (*StripePercentField*) – A positive decimal (with at most two decimal places) between 1 and 100. This represents the percentage of the subscription invoice subtotal that will be calculated and added as tax to the final amount each billing period.
- **trial_end** (*StripeDateTimeField*) – If the subscription has a trial, the end of that trial.
- **trial_start** (*StripeDateTimeField*) – If the subscription has a trial, the beginning of that trial.

api_retrieve(*api_key=None*)

Call the stripe API's retrieve operation for this model.

Parameters **api_key** (*string*) – The api key to use for this request. Defaults to settings.STRIPE_SECRET_KEY.

is_period_current()

Returns True if this subscription's period is current, false otherwise.

is_status_current()
Returns True if this subscription's status is current (active or trialing), false otherwise.

is_status_temporarily_current()
A status is temporarily current when the subscription is canceled with the at_period_end flag. The subscription is still active, but is technically canceled and we're just waiting for it to run out.

You could use this method to give customers limited service after they've canceled. For example, a video on demand service could only allow customers to download their libraries and do nothing else when their subscription is temporarily current.

is_valid()
Returns True if this subscription's status and period are current, false otherwise.

update(*plan=None*, *application_fee_percent=None*, *coupon=None*, *prorate=False*, *proration_date=None*, *metadata=None*, *quantity=None*, *tax_percent=None*, *trial_end=None*)
See *Customer.subscribe()*

Parameters

- **plan** (*Plan or string (plan ID)*) – The plan to which to subscribe the customer.
- **prorate** (*boolean*) – Whether or not to prorate when switching plans. Default is True.
- **proration_date** (*datetime*) – If set, the proration will be calculated as though the subscription was updated at the given time. This can be used to apply exactly the same proration that was previewed with upcoming invoice endpoint. It can also be used to implement custom proration logic, such as prorating by day instead of by second, by providing the time that you wish to use for proration calculations.

Note: The default value for prorate is the DJSTRIPE_PRORATION_POLICY setting.

Important: Updating a subscription by changing the plan or quantity creates a new Subscription in Stripe (and dj-stripe).

---

extend(*delta*)
Extends this subscription by the provided delta.

Parameters **delta** (*timedelta*) – The timedelta by which to extend this subscription.

cancel(*at_period_end=True*)
Cancels this subscription. If you set the at_period_end parameter to true, the subscription will remain active until the end of the period, at which point it will be canceled and not renewed. By default, the subscription is terminated immediately. In either case, the customer will not be charged again for the subscription. Note, however, that any pending invoice items that you've created will still be charged for at the end of the period unless manually deleted. If you've set the subscription to cancel at period end, any pending prorations will also be left in place and collected at the end of the period, but if the subscription is set to cancel immediately, pending prorations will be removed.

By default, all unpaid invoices for the customer will be closed upon subscription cancellation. We do this in order to prevent unexpected payment retries once the customer has canceled a subscription. However, you can reopen the invoices manually after subscription cancellation to have us proceed with automatic retries, or you could even re-attempt payment yourself on all unpaid invoices before allowing the customer to cancel the subscription at all.

Parameters **at_period_end** (*boolean*) – A flag that if set to true will delay the cancellation of the subscription until the end of the current period. Default is False.

Important: If a subscription is cancelled during a trial period, the `at_period_end` flag will be overridden to False so that the trial ends immediately and the customer's card isn't charged.

str_parts()
Extend this to add information to the string representation of the object

Return type list of str

1.4 Settings

1.4.1 STRIPE_API_VERSION (='2017-02-14')

The API version used to communicate with the Stripe API is configurable, and defaults to the latest version that has been tested as working. Using a value other than the default is allowed, as a string in the format of YYYY-MM-DD.

For example, you can specify *'2017-01-27'* to use that API version:

```
STRIPE_API_VERSION = '2017-01-27'
```

However you do so at your own risk, as using a value other than the default might result in incompatibilities between Stripe and this library, especially if Stripe has labelled the differences between API versions as "Major". Even small differences such as a new enumeration value might cause issues.

For this reason it is best to assume that only the default version is supported.

For more information on API versioning, see the **'stripe documentation'_**.

1.4.2 DJSTRIPE_IDEMPOTENCY_KEY_CALLBACK (=djstripe.settings._get_idempotency_key)

A function which will return an idempotency key for a particular object_type and action pair. By default, this is set to a function which will create a `djstripe.IdempotencyKey` object and return its `uuid`. You may want to customize this if you want to give your idempotency keys a different lifecycle than they normally would get.

The function takes the following signature:

```
def get_idempotency_key(object_type: str, action: str, livemode: bool):
    return "<idempotency key>"
```

The function MUST return a string suitably random for the object_type/action pair, and usable in the Stripe `Idempotency-Key` HTTP header. For more information, see the **'stripe documentation'_**.

1.4.3 DJSTRIPE_PRORATION_POLICY (=False)

By default, plans are not prorated in dj-stripe. Concretely, this is how this translates:

1. If a customer cancels their plan during a trial, the cancellation is effective right away.
2. If a customer cancels their plan outside of a trial, their subscription remains active until the subscription's period end, and they do not receive a refund.
3. If a customer switches from one plan to another, the new plan becomes effective right away, and the customer is billed for the new plan's amount.

Assigning `True` to `DJSTRIPE_PRORATION_POLICY` reverses the functioning of item 2 (plan cancellation) by making a cancellation effective right away and refunding the unused balance to the customer, and affects the functioning of item 3 (plan change) by prorating the previous customer's plan towards their new plan's amount.

1.4.4 DJSTRIPE_SUBSCRIPTION_REQUIRED_EXCEPTION_URLS (=())

Used by `djstripe.middleware.SubscriptionPaymentMiddleware`

Rules:

- "(app_name)" means everything from this app is exempt
- "[namespace]" means everything with this name is exempt
- "namespace:name" means this namespaced URL is exempt
- "name" means this URL is exempt
- The entire djstripe namespace is exempt
- If settings.DEBUG is True, then django-debug-toolbar is exempt

Example:

```
DJSTRIPE_SUBSCRIPTION_REQUIRED_EXCEPTION_URLS = (
    "(allauth)",  # anything in the django-allauth URLConf
    "[blogs]",  # Anything in the blogs namespace
    "products:detail",  # A ProductDetail view you want shown to non-payers
    "home",  # Site homepage
)
```

Note: Adding app_names to applications.

To make the `(allauth)` work, you may need to define an app_name in the `include()` function in the URLConf. For example:

```
# in urls.py
url(r'^accounts/', include('allauth.urls',  app_name="allauth")),
```

1.4.5 DJSTRIPE_SUBSCRIBER_MODEL (=settings.AUTH_USER_MODEL)

If the AUTH_USER_MODEL doesn't represent the object your application's subscription holder, you may define a subscriber model to use here. It should be a string in the form of 'app.model'.

Rules:

- DJSTRIPE_SUBSCRIBER_MODEL must have an `email` field. If your existing model has no email field, add an email property that defines an email address to use.
- You must also implement `DJSTRIPE_SUBSCRIBER_MODEL_REQUEST_CALLBACK`.

Example Model:

```
class Organization(models.Model):
    name = CharField(max_length=200, unique=True)
    subdomain = CharField(max_length=63, unique=True, verbose_name="Organization␣
↪Subdomain")
    owner = ForeignKey(settings.AUTH_USER_MODEL, related_name="organization_owner",␣
↪verbose_name="Organization Owner")
```

```
    @property
    def email(self):
        return self.owner.email
```

1.4.6 DJSTRIPE_SUBSCRIBER_MODEL_MIGRATION_DEPENDENCY (="__first__")

If the model referenced in DJSTRIPE_SUBSCRIBER_MODEL is not created in the `__first__` migration of an app you can specify the migration name to depend on here. For example: "0003_here_the_subscriber_model_was_added"

1.4.7 DJSTRIPE_SUBSCRIBER_MODEL_REQUEST_CALLBACK (=None)

If you choose to use a custom subscriber model, you'll need a way to pull it from `request`. That's where this callback comes in. It must be a callable or importable string to a callable that takes a request object and returns an instance of DJSTRIPE_SUBSCRIBER_MODEL

Examples:

middleware.py

```
class DynamicOrganizationIDMiddleware(object):
    """ Adds the current organization's ID based on the subdomain."""

    def process_request(self, request):
        subdomain = parse_subdomain(request.get_host())

        try:
            organization = Organization.objects.get(subdomain=subdomain)
        except Organization.DoesNotExist:
            return TemplateResponse(request=request, template='404.html', status=404)
        else:
            organization_id = organization.id

        request.organization_id = organization_id
```

settings.py

```
def organization_request_callback(request):
    """ Gets an organization instance from the id passed through ``request``"""

    from <models_path> import Organization  # Import models here to avoid an␣
↪``AppRegistryNotReady`` exception
    return Organization.objects.get(id=request.organization_id)
```

Note: This callback only becomes active when `DJSTRIPE_SUBSCRIBER_MODEL` is set.

1.4.8 DJSTRIPE_USE_NATIVE_JSONFIELD (=False)

Setting this to `True` will make the various dj-stripe JSON fields use `django.contrib.postgres.fields.JSONField` instead of the `jsonfield` library (which internally uses `text` fields).

The native Django JSONField uses the postgres jsonb column type, which efficiently stores JSON and can be queried far more conveniently. Django also supports querying JSONField with the ORM.

Note: This is only supported on Postgres databases.

Note: **Migrating between native and non-native must be done manually.**

1.4.9 DJSTRIPE_WEBHOOK_URL (=r"^webhook/$")

This is where you can set *Stripe.com* to send webhook response. You can set this to what you want to prevent unnecessary hijinks from unfriendly people.

As this is embedded in the URLConf, this must be a resolvable regular expression.

1.4.10 DJSTRIPE_WEBHOOK_EVENT_CALLBACK (=None)

Webhook event callbacks allow an application to take control of what happens when an event from Stripe is received. It must be a callable or importable string to a callable that takes an event object.

One suggestion is to put the event onto a task queue (such as celery) for asynchronous processing.

Examples:

callbacks.py

```
def webhook_event_callback(event):
    """ Dispatches the event to celery for processing. """
    from . import tasks
    # Ansychronous hand-off to celery so that we can continue immediately
    tasks.process_webhook_event.s(event).apply_async()
```

tasks.py

```
from stripe.error import StripeError

@shared_task(bind=True)
def process_webhook_event(self, event):
    """ Processes events from Stripe asynchronously. """
    logger.info("Processing Stripe event: %s", str(event))
    try:
        event.process(raise_exception=True)
    except StripeError as exc:
        logger.error("Failed to process Stripe event: %s", str(event))
        raise self.retry(exc=exc, countdown=60)  # retry after 60 seconds
```

settings.py

```
DJSTRIPE_WEBHOOK_EVENT_CALLBACK = 'callbacks.webhook_event_callback'
```

1.5 Cookbook

This is a list of handy recipes that fall outside the domain of normal usage.

1.5.1 Customer User Model has_active_subscription property

Very useful for working inside of templates or other places where you need to check the subscription status repeatedly. The *cached_property* decorator caches the result of *has_active_subscription* for a object instance, optimizing it for reuse.

```
# -*- coding: utf-8 -*-

from django.contrib.auth.models import AbstractUser
from django.db import models
from django.utils.functional import cached_property

from djstripe.utils import subscriber_has_active_subscription


class User(AbstractUser):

    """ Custom fields go here """

    def __str__(self):
        return self.username

    def __unicode__(self):
        return self.username

    @cached_property
    def has_active_subscription(self):
        """Checks if a user has an active subscription."""
        return subscriber_has_active_subscription(self)
```

Usage:

```
<ul class="actions">
<h2>{{ object }}</h2>
<!-- first use of request.user.has_active_subscription -->
{% if request.user.has_active_subscription %}
    <p>
        <small>
            <a href="{% url 'things:update' %}">edit</a>
        </small>
    </p>
{% endif %}
<p>{{ object.description }}</p>

<!-- second use of request.user.has_active_subscription -->
{% if request.user.has_active_subscription %}
    <li>
        <a href="{% url 'places:create' %}">Add Place</a>
        <a href="{% url 'places:list' %}">View Places</a>
    </li>
{% endif %}
</ul>
```

1.5.2 Making individual purchases

On the subscriber's customer object, use the charge method to generate a Stripe charge. In this example, we're using the user with ID=1 as the subscriber.

```
from decimal import Decimal

from django.contrib.auth import get_user_model

from djstripe.models import Customer


user = get_user_model().objects.get(id=1)

customer, created = Customer.get_or_create(subscriber=user)

amount = Decimal(10.00)
customer.charge(amount)
```

Source code for the Customer.charge method is at https://github.com/dj-stripe/dj-stripe/blob/master/djstripe/models.py

1.5.3 REST API

The subscriptions can be accessed through a REST API. Make sure you have Django Rest Framework installed (https://github.com/tomchristie/django-rest-framework).

The REST API endpoints require an authenticated user. GET will provide the current subscription of the user. POST will create a new current subscription. DELETE will cancel the current subscription, based on the settings.

- **/subscription/ (GET)**
  - **input**
    - None
  - output (200)
    - id (int)
    - created (date)
    - modified (date)
    - plan (string)
    - quantity (int)
    - start (date)
    - status (string)
    - cancel_at_period_end (boolean)
    - canceled_at (date)
    - current_period_end (date)
    - current_period_start (date)
    - ended_at (date)
    - trial_end (date)
    - trial_start (date)

        * amount (float)
        * customer (int)

- **/subscription/ (POST)**
  - **input**
    * stripe_token (string)
    * plan (string)
    * charge_immediately (boolean, optional) - Does not send an invoice to the Customer immediately
  - **output (201)**
    * stripe_token (string)
    * plan (string)
- **/subscription/ (DELETE)**
  - **input**
    * None
  - **Output (204)**
    * None

1.6 Not in the Cookbook?

Cartwheel Web provides commercial support for dj-stripe and other open source packages.

1.7 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

1.7.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/dj-stripe/dj-stripe/issues.

If you are reporting a bug, please include:

- The version of python and Django you're running
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

dj-stripe could always use more documentation, whether as part of the official dj-stripe docs, in docstrings, or even on the web in blog posts, articles, and such.

If you are adding to dj-stripe's documentation, you can see your changes by changing into the `docs` directory, running `make html` (or `make.bat html` if you're developing on Windows) from the command line, and then opening `docs/_build/html/index.html` in a web browser.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/dj-stripe/dj-stripe/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

1.7.2 New Modules

As with Django we're aiming for future compatibility with Python 3.x. Please ensure that any new modules use the following future import statement:

```
` from __future__ import absolute_import, division, print_function,
unicode_literals `
```

1.7.3 Get Started!

Ready to contribute? Here's how to set up *dj-stripe* for local development.

1. Fork the *dj-stripe* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/dj-stripe.git
```

3. Assuming the tests are run against PostgreSQL:

```
$ createdb djstripe
```

4. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv dj-stripe
$ cd dj-stripe/
$ python setup.py develop
```

5. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

6. When you're done making changes, check that your changes pass the tests, including testing other Python versions with tox. runtests will output both command line and html coverage statistics and will warn you if your changes caused code coverage to drop. Note that if your system time is not in UTC, some tests will fail. If you want to ignore those tests, the –skip-utc command line option is available on runtests.py.:

```
$ pip install -r tests/requirements.txt
$ tox
```

7. If your changes altered the models you may need to generate Django migrations:

```
$ python makemigrations.py
```

8. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

9. Submit a pull request through the GitHub website.
10. Congratulations, you're now a dj-stripe contributor! Have some <3 from us.

1.7.4 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. The pull request must not drop code coverage below the current level.
3. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
4. If the pull request makes changes to a model, include Django migrations (Django 1.7+).
5. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6. Check https://travis-ci.org/dj-stripe/dj-stripe/pull_requests and make sure that the tests pass for all supported Python versions.

1.8 Credits

1.8.1 Development Lead

- Alexander Kavanaugh (@kavdev)
- Daniel Greenfeld <pydanny@gmail.com>
- Jerome Leclanche (@jleclanche)
- Lee Skillen (@lskillen)

1.8.2 Contributors

- Audrey Roy Greenfeld (@audreyr)
- Buddy Lindsley (@buddylindsey)
- Yasmine Charif (@dollydagr)
- Mahdi Yusuf (@myusuf3)
- Luis Montiel <luismmontielg@gmail.com>
- Kulbir Singh (@kulbir)
- Dustin Farris (@dustinfarris)
- Liwen S (@sunliwen)
- centrove
- Chris Halpert (@cphalpert)
- Thomas Parslow (@almost)
- Leonid Shvechikov (@shvechikov)
- sromero84
- Peter Baumgartner (@ipmb)
- Vikas (@vikasgulati)
- Colton Allen (@cmanallen)
- Filip Wasilewski (@nigma)
- Martin Hill (@martinhill)
- Michael Thornhill <michael.thornhill@gmail.com>
- Tobias Lorenz (@Tyrdall)
- Ben Whalley
- nanvel
- jRobb (@jamesbrobb)
- Areski Belaid (@areski)
- José Padilla (@jpadilla)
- Ben Murden (@benmurden)
- Philippe Luickx (@philippeluickx)
- Chriss Mejía (@chrissmejia)
- Bill Huneke (@wahuneke)
- Matt Shaw (@unformatt)
- Chris Trengove (@ctrengove)
- Caleb Hattingh (@cjrh)
- Nicolas Delaby (@ticosax)
- Michaël Krens (@michi88)
- Yuri Prezument (@yprez)

- Raphael Deem (@r0fls)
- Irfan Ahmad (@erfaan)
- Slava Kyrachevsky (@scream4ik)
- Alec Brunelle (@aleccool213)
- James Hiew (@jameshiew)
- Dan Koch (@dmkoch)

1.9 History

1.9.1 1.0.0 (2017-08-12)

It's finally here! We've made significant changes to the codebase and are now compliant with stripe API version **2017-06-05**.

I want to give a huge thanks to all of our contributors for their help in making this happen, especially Bill Huneke (@wahuneke) for his impressive design work and @jleclanche for really pushing this release along.

I also want to welcome onboard two more maintainers, @jleclanche and @lskillen. They've stepped up and have graciously dedicated their resources to making dj-stripe such an amazing package.

Almost all methods now mimic the parameters of those same methods in the stripe API. Note that some methods do not have some parameters implemented. This is intentional. That being said, expect all method signatures to be different than those in previous versions of dj-stripe.

Finally, please note that there is still a bit of work ahead of us. Not everything in the Stripe API is currently supported by dj-stripe – we're working on it. That said, v1.0.0 has been thoroughly tested and is verified stable in production applications.

A few things to get excited for

- Multiple subscription support (finally)
- Multiple sources support (currently limited to Cards)
- Idempotency support (See #455, #460 for discussion – big thanks to @jleclanche)
- Full model documentation
- Objects that come through webhooks are now tied to the API version set in dj-stripe. No more errors if dj-stripe falls behind the newest stripe API version.
- Any create/update action on an object automatically syncs the object.
- Concurrent LIVE and TEST mode support (Thanks to @jleclanche). Note that you'll run into issues if `livemode` isn't set on your existing customer objects.
- All choices are now enum-based (Thanks @jleclanche, See #520). Access them from the new `djstripe.enums` module. The ability to check against model property based choices will be deprecated in 1.1
- Support for the Coupon model, and coupons on Customer objects.
- Support for the Payout/Transfer split from api version `2017-04-06`.

What still needs to be done (in v1.1.0)

- **Documentation**. Our original documentation was not very helpful, but it covered the important bits. It will be very out of date after this update and will need to be rewritten. If you feel like helping, we could use all the help we can get to get this pushed out asap.
- **Master sync re-write**. This sounds scary, but really isn't. The current management methods run sync methods on Customer that aren't very helpful and are due for removal. My plan is to write something that first updates local data (via `api_retrieve` and `sync_from_stripe_data`) and then pulls all objects from Stripe and populates the local database with any records that don't already exist there.

  You might be wondering, "Why are they releasing this if there are only a few things left?" Well, that thinking turned this into a two year release... Trust me, this is a good thing.

Significant changes (mostly backwards-incompatible)

- **Idempotency**. #460 introduces idempotency keys and implements idempotency for `Customer.get_or_create()`. Idempotency will be enabled for all calls that need it.
- **Improved Admin Interface**. This is almost complete. See #451 and #452.
- **Drop non-trivial endpoint views**. We're dropping everything except the webhook endpoint and the subscription cancel endpoint. See #428.
- **Drop support for sending receipts**. Stripe now handles this for you. See #478.
- **Drop support for plans as settings**, including custom plan hierarchy (if you want this, write something custom) and the dynamic trial callback. We've decided to gut having plans as settings. Stripe should be your source of truth; create your plans there and sync them down manually. If you need to create plans locally for testing, etc., simply use the ORM to create Plan models. The sync rewrite will make this drop less annoying.
- **Orphan Customer Sync**. We will now sync Customer objects from Stripe even if they aren't linked to local subscriber objects. You can link up subscribers to those Customers manually.
- **Concurrent Live and Test Mode**. dj-stripe now supports test-mode and live-mode Customer objects concurrently. As a result, the User.customer One-to-One reverse-relationship is now the User.djstripe_customers RelatedManager. (Thanks @jleclanche) #440. You'll run into some dj-stripe check issues if you don't update your KEY settings accordingly. Check our GitHub issue tracker for help on this.

SETTINGS

- The `PLAN_CHOICES`, `PLAN_LIST`, and `PAYMENT_PLANS` objects are removed. Use Plan.objects.all() instead.
- The `plan_from_stripe_id` function is removed. Use Plan.objects.get(stripe_id=)

SYNCING

- sync_plans no longer takes an api_key
- sync methods no longer take a `cu` parameter
- All sync methods are now private. We're in the process of building a better syncing mechanism.

UTILITIES

- dj-stripe decorators now take a plan argument. If you're passing in a custom test function to `subscriber_passes_pay_test`, be sure to account for this new argument.

MIXINS

- The context provided by dj-stripe's mixins has changed. `PaymentsContextMixin` now provides `STRIPE_PUBLIC_KEY` and `plans` (changed to `Plan.objects.all()`). `SubscriptionMixin` now provides `customer` and `is_plans_plural`.
- We've removed the SubscriptionPaymentRequiredMixin. Use `@method_decorator("dispatch", subscription_payment_required)` instead.

MIDDLEWARE

- dj-stripe middleware doesn't support multiple subscriptions.

SIGNALS

- Local custom signals are deprecated in favor of Stripe webhooks:
- `cancelled` -> WEBHOOK_SIGNALS["customer.subscription.deleted"]
- `card_changed` -> WEBHOOK_SIGNALS["customer.source.updated"]
- `subscription_made` -> WEBHOOK_SIGNALS["customer.subscription.created"]

WEBHOOK EVENTS

- The Event Handlers designed by @wahuneke are the new way to handle events that come through webhooks. Definitely take a look at `event_handlers.py` and `webhooks.py`.

EXCEPTIONS

- `SubscriptionUpdateFailure` and `SubscriptionCancellationFailure` exceptions are removed. There should no longer be a case where they would have been useful. Catch native stripe errors in their place instead.

MODELS

CHARGE

- `Charge.charge_created` -> `Charge.stripe_timestamp`
- `Charge.card_last_4` and `Charge.card_kind` are removed. Use `Charge.source.last4` and `Charge.source.brand` (if the source is a Card)
- `Charge.invoice` is no longer a foreign key to the Invoice model. `Invoice` now has a OneToOne relationship with `Charge`. (`Charge.invoice` will still work, but will no longer be represented in the database).

CUSTOMER

- dj-stripe now supports test mode and live mode Customer objects concurrently (See #440). As a result, the `<subscriber_model>.customer` OneToOne reverse relationship is no longer a thing. You should now instead add a `customer` property to your subscriber model that checks whether you're in live or test mode (see djstripe.settings.STRIPE_LIVE_MODE as an example) and grabs the customer from `<subscriber_model>.djstripe_customers` with a simple `livemode=` filter.
- Customer no longer has a `current_subscription` property. We've added a `subscription` property that should suit your needs.
- With the advent of multiple subscriptions, the behavior of `Customer.subscribe()` has changed. Before, `calling subscribe()` when a customer was already subscribed to a plan would switch the customer to the new plan with an option to prorate. Now calling `subscribe()` simply subscribes that customer to a new plan in addition to it's current subsription. Use `Subscription.update()` to change a subscription's plan instead.
- `Customer.cancel_subscription()` is removed. Use `Subscription.cancel()` instead.
- The `Customer.update_plan_quantity()` method is removed. Use `Subscription.update()` instead.
- `CustomerManager` is now `SubscriptionManager` and works on the `Subscription` model instead of the `Customer` model.
- `Customer.has_valid_card()` is now `Customer.has_valid_source()`.
- `Customer.update_card()` now takes an id. If the id is not supplied, the default source is updated.
- `Customer.stripe_customer` property is removed. Use `Customer.api_retrieve()` instead.
- The `at_period_end` parameter of `Customer.cancel_subscription()` now actually follows the DJSTRIPE_PRORATION_POLICY setting.
- `Customer.card_fingerprint`, `Customer.card_last_4`, `Customer.card_kind`, `Customer.card_exp_month`, `Customer.card_exp_year` are all removed. Check `Customer.default_source` (if it's a Card) or one of the sources in `Customer.sources` (again, if it's a Card) instead.
- The `invoice_id` parameter of `Customer.add_invoice_item` is now named `invoice` and can be either an Invoice object or the stripe_id of an Invoice.

EVENT

- `Event.kind` -> `Event.type`
- Removed `Event.validated_message`. Just check if the event is valid - no need to double check (we do that for you)

TRANSFER

- Removed `Transfer.update_status()`
- Removed `Transfer.event`
- `TransferChargeFee` is removed. It hasn't been used in a while due to a broken API version. Use `Transfer.fee_details` instead.

- Any fields that were in `Transfer.summary` no longer exist and are therefore deprecated (unused but not removed from the database). Because of this, `TransferManager` now only aggregates `total_sum`

INVOICE

- `Invoice.attempts` -> `Invoice.attempt_count`
- InvoiceItems are no longer created when Invoices are synced. You must now sync InvoiceItems directly.

INVOICEITEM

- Removed `InvoiceItem.line_type`

PLAN

- Plan no longer has a `stripe_plan` property. Use `api_retrieve()` instead.
- `Plan.currency` no longer uses choices. Use the `get_supported_currency_choices()` utility and create your own custom choices list instead.
- Plan interval choices are now in `Plan.INTERVAL_TYPE_CHOICES`

SUBSCRIPTION

- `Subscription.is_period_current()` now checks for a current trial end if the current period has ended. This change means subscriptions extended with `Subscription.extend()` will now be seen as valid.

MIGRATIONS

We'll sync your current records with Stripe in a migration. It will take a while, but it's the only way we can ensure data integrity. There were some fields for which we needed to temporarily add placeholder defaults, so just make sure you have a customer with ID 1 and a plan with ID 1 and you shouldn't run into any issues (create dummy values for these if need be and delete them after the migration).

BIG HUGE NOTE - DON'T OVERLOOK THIS

Subscription and InvoiceItem migration is not possible because old records don't have Stripe IDs (so we can't sync them). Our approach is to delete all local subscription and invoiceitem objects and re-sync them from Stripe.

We 100% recommend you create a backup of your database before performing this upgrade.

Other changes

- Postgres users now have access to the `DJSTRIPE_USE_NATIVE_JSONFIELD` setting. (Thanks @jleclanche) #517, #523
- Charge receipts now take `DJSTRIPE_SEND_INVOICE_RECEIPT_EMAILS` into account (Thanks @r0fls)

- Clarified/modified installation documentation (Thanks @pydanny)
- Corrected and revised ANONYMOUS_USER_ERROR_MSG (Thanks @pydanny)
- Added fnmatching to `SubscriptionPaymentMiddleware` (Thanks @pydanny)
- `SubscriptionPaymentMiddleware.process_request()` functionality broken up into multiple methods, making local customizations easier (Thanks @pydanny)
- Fully qualified events are now supported by event handlers as strings e.g. 'customer.subscription.deleted' (Thanks @lskillen) #316
- runtests now accepts positional arguments for declaring which tests to run (Thanks @lskillen) #317
- It is now possible to reprocess events in both code and the admin interface (Thanks @lskillen) #318
- The confirm page now checks that a valid card exists. (Thanks @scream4ik) #325
- Added support for viewing upcoming invoices (Thanks @lskillen) #320
- Event handler improvements and bugfixes (Thanks @lskillen) #321
- API list() method bugfixes (Thanks @lskillen) #322
- Added support for a custom webhook event handler (Thanks @lskillen) #323
- Django REST Framework contrib package improvements (Thanks @aleccool213) #334
- Added `tax_percent` to CreateSubscriptionSerializer (Thanks @aleccool213) #349
- Fixed incorrectly assigned `application_fee` in Charge calls (Thanks @kronok) #382
- Fixed bug caused by API change (Thanks @jessamynsmith) #353
- Added inline documentation to pretty much everything and enforced docsytle via flake8 (Thanks @aleccool213)
- Fixed outdated method call in template (Thanks @kandoio) #391
- Customer is correctly purged when subscriber is deleted, regardless of how the deletion happened (Thanks @lskillen) #396
- Test webhooks are now properly captured and logged. No more bounced requests to Stripe! (Thanks @jameshiew) #408
- CancelSubscriptionView redirect is now more flexible (Thanks @jleclanche) #418
- Customer.sync_cards() (Thanks @jleclanche) #438
- Many stability fixes, bugfixes, and code cleanup (Thanks @jleclanche)
- Support syncing cancelled subscriptions (Thanks @jleclanche) #443
- Improved admin interface (Thanks @jleclanche with @jameshiew) #451
- Support concurrent TEST + LIVE API keys (Fix webhook event processing for both modes) (Thanks @jleclanche) #461
- Added Stripe Dashboard link to admin change panel (Thanks @jleclanche) #465
- Implemented `Plan.amount_in_cents` (Thanks @jleclanche) #466
- Implemented `Subscription.reactivate()` (Thanks @jleclanche) #470
- Added `Plan.human_readable_price` (Thanks @jleclanche) #498
- (Re)attach the Subscriber when we find it's id attached to a customer on Customer sync (Thanks @jleclanche) #500
- Made API version configurable (with dj-stripe recommended default) (Thanks @lskillen) #504

1.9.2 0.8.0 (2015-12-30)

- better plan ordering documentation (Thanks @cjrh)
- added a confirmation page when choosing a subscription (Thanks @chrissmejia, @areski)
- setup.py reverse dependency fix (#258/#268) (Thanks @ticosax)
- Dropped official support for Django 1.7 (no code changes were made)
- Python 3.5 support, Django 1.9.1 support
- Migration improvements (Thanks @michi88)
- Fixed "Invoice matching query does not exist" bug (#263) (Thanks @mthornhill)
- Fixed duplicate content in account view (Thanks @areski)

1.9.3 0.7.0 (2015-09-22)

- dj-stripe now responds to the invoice.created event (Thanks @wahuneke)
- dj-stripe now cancels subscriptions and purges customers during sync if they were deleted from the stripe dashboard (Thanks @unformatt)
- dj-stripe now checks for an active stripe subscription in the `update_plan_quantity` call (Thanks @ctrengove)
- Event processing is now handled by "event handlers" - functions outside of models that respond to various event types and subtypes. Documentation on how to tie into the event handler system coming soon. (Thanks @wahuneke)
- Experimental Python 3.5 support
- Support for Django 1.6 and lower is now officially gone.
- Much, much more!

1.9.4 0.6.0 (2015-07-12)

- Support for Django 1.6 and lower is now deprecated.
- Improved test harness now tests coverage and pep8
- SubscribeFormView and ChangePlanView no longer populate self.error with form errors
- InvoiceItems.plan can now be null (as it is with individual charges), resolving #140 (Thanks @awechsler and @MichelleGlauser for help troubleshooting)
- Email templates are now packaged during distribution.
- sync_plans now takes an optional api_key
- 100% test coverage
- Stripe ID is now returned as part of each model's str method (Thanks @areski)
- Customer model now stores card expiration month and year (Thanks @jpadilla)
- Ability to extend subscriptions (Thanks @TigerDX)
- Support for plan heirarchies (Thanks @chrissmejia)
- Rest API endpoints for Subscriptions [contrib] (Thanks @philippeluickx)

- Admin interface search by email funtionality is removed (#221) (Thanks @jpadilla)

1.9.5 0.5.0 (2015-05-25)

- Began deprecation of support for Django 1.6 and lower.
- Added formal support for Django 1.8.
- Removed the StripeSubscriptionSignupForm
- Removed `djstripe.safe_settings`. Settings are now all located in `djstripe.settings`
- `DJSTRIPE_TRIAL_PERIOD_FOR_SUBSCRIBER_CALLBACK` can no longer be a module string
- The sync_subscriber argument has been renamed from subscriber_model to subscriber
- Moved available currencies to the DJSTRIPE_CURRENCIES setting (Thanks @martinhill)
- Allow passing of extra parameters to stripe Charge API (Thanks @mthornhill)
- Support for all available arguments when syncing plans (Thanks @jamesbrobb)
- charge.refund() now returns the refunded charge object (Thanks @mthornhill)
- Charge model now has captured field and a capture method (Thanks @mthornhill)
- Subscription deleted webhook bugfix
- South migrations are now up to date (Thanks @Tyrdall)

1.9.6 0.4.0 (2015-04-05)

- Formal Python 3.3+/Django 1.7 Support (including migrations)
- Removed Python 2.6 from Travis CI build. (Thanks @audreyr)
- Dropped Django 1.4 support. (Thanks @audreyr)
- Deprecated the `djstripe.forms.StripeSubscriptionSignupForm`. Making this form work easily with both `dj-stripe` and `django-allauth` required too much abstraction. It will be removed in the 0.5.0 release.
- Add the ability to add invoice items for a customer (Thanks @kavdev)
- Add the ability to use a custom customer model (Thanks @kavdev)
- Added setting to disable Invoice receipt emails (Thanks Chris Halpert)
- Enable proration when customer upgrades plan, and pass proration policy and cancellation at period end for upgrades in settings. (Thanks Yasmine Charif)
- Removed the redundant context processor. (Thanks @kavdev)
- Fixed create a token call in change_card.html (Thanks @dollydagr)
- Fix `charge.dispute.closed` typo. (Thanks @ipmb)
- Fix contributing docs formatting. (Thanks @audreyr)
- Fix subscription cancelled_at_period_end field sync on plan upgrade (Thanks @nigma)
- Remove "account" bug in Middleware (Thanks @sromero84)
- Fix correct plan selection on subscription in subscribe_form template. (Thanks Yasmine Charif)

- Fix subscription status in account, _subscription_status, and cancel_subscription templates. (Thanks Yasmine Charif)
- Now using `user.get_username()` instead of `user.username`, to support custom User models. (Thanks @shvechikov)
- Update remaining DOM Ids for Bootstrap 3. (Thanks Yasmine Charif)
- Update publish command in setup.py. (Thanks @pydanny)
- Explicitly specify tox's virtual environment names. (Thanks @audreyr)
- Manually call django.setup() to populate apps registry. (Thanks @audreyr)

1.9.7 0.3.5 (2014-05-01)

- Fixed `djstripe_init_customers` management command so it works with custom user models.

1.9.8 0.3.4 (2014-05-01)

- Clarify documentation for redirects on app_name.
- If settings.DEBUG is True, then django-debug-toolbar is exempt from redirect to subscription form.
- Use collections.OrderedDict to ensure that plans are listed in order of price.
- Add `ordereddict` library to support Python 2.6 users.
- Switch from `__unicode__` to `__str__` methods on models to better support Python 3.
- Add `python_2_unicode_compatible` decorator to Models.
- Check for PY3 so the `unicode(self.user)` in models.Customer doesn't blow up in Python 3.

1.9.9 0.3.3 (2014-04-24)

- Increased the extendability of the views by removing as many hard-coded URLs as possible and replacing them with `success_url` and other attributes/methods.
- Added single unit purchasing to the cookbook

1.9.10 0.3.2 (2014-01-16)

- Made Yasmine Charif a core committer
- Take into account trial days in a subscription plan (Thanks Yasmine Charif)
- Correct invoice period end value (Thanks Yasmine Charif)
- Make plan cancellation and plan change consistently not prorating (Thanks Yasmine Charif)
- Fix circular import when ACCOUNT_SIGNUP_FORM_CLASS is defined (Thanks Dustin Farris)
- Add send e-mail receipt action in charges admin panel (Thanks Buddy Lindsay)
- Add `created` field to all ModelAdmins to help with internal auditing (Thanks Kulbir Singh)

1.9.11 0.3.1 (2013-11-14)

- Cancellation fix (Thanks Yasmine Charif)
- Add setup.cfg for wheel generation (Thanks Charlie Denton)

1.9.12 0.3.0 (2013-11-12)

- Fully tested against Django 1.6, 1.5, and 1.4
- Fix boolean default issue in models (from now on they are all default to `False`).
- Replace duplicated code with `djstripe.utils.user_has_active_subscription`.

1.9.13 0.2.9 (2013-09-06)

- Cancellation added to views.
- Support for kwargs on charge and invoice fetching.
- def charge() now supports send_receipt flag, default to True.
- Fixed templates to work with Bootstrap 3.0.0 column design.

1.9.14 0.2.8 (2013-09-02)

- Improved usage documentation.
- Corrected order of fields in StripeSubscriptionSignupForm.
- Corrected transaction history template layout.
- Updated models to take into account when settings.USE_TZ is disabled.

1.9.15 0.2.7 (2013-08-24)

- Add handy rest_framework permission class.
- Fixing attribution for django-stripe-payments.
- Add new status to Invoice model.

1.9.16 0.2.6 (2013-08-20)

- Changed name of division tag to djdiv.
- Added `safe_setting.py` module to handle edge cases when working with custom user models.
- Added cookbook page in the documentation.

1.9.17 0.2.5 (2013-08-18)

- Fixed bug in initial checkout
- You can't purchase the same plan that you currently have.

1.9.18 0.2.4 (2013-08-18)

- Recursive package finding.

1.9.19 0.2.3 (2013-08-16)

- Fix packaging so all submodules are loaded

1.9.20 0.2.2 (2013-08-15)

- Added Registration + Subscription form

1.9.21 0.2.1 (2013-08-12)

- Fixed a bug on CurrentSubscription tests
- Improved usage documentation
- Added to migration from other tools documentation

1.9.22 0.2.0 (2013-08-12)

- Cancellation of plans now works.
- Upgrades and downgrades of plans now work.
- Changing of cards now works.
- Added breadcrumbs to improve navigation.
- Improved installation instructions.
- Consolidation of test instructions.
- Minor improvement to django-stripe-payments documentation
- Added coverage.py to test process.
- Added south migrations.
- Fixed the subscription_payment_required function-based view decorator.
- Removed unnecessary django-crispy-forms

1.9.23 0.1.7 (2013-08-08)

- Middleware excepts all of the djstripe namespaced URLs. This way people can pay.

1.9.24 0.1.6 (2013-08-08)

- Fixed a couple template paths
- Fixed the manifest so we include html, images.

1.9.25 0.1.5 (2013-08-08)

- Fixed the manifest so we include html, css, js, images.

1.9.26 0.1.4 (2013-08-08)

- Change PaymentRequiredMixin to SubscriptionPaymentRequiredMixin
- Add subscription_payment_required function-based view decorator
- Added SubscriptionPaymentRedirectMiddleware
- Much nicer accounts view display
- Much improved subscription form display
- Payment plans can have decimals
- Payment plans can have custom images

1.9.27 0.1.3 (2013-08-7)

- Added account view
- Added Customer.get_or_create method
- Added djstripe_sync_customers management command
- sync file for all code that keeps things in sync with stripe
- Use client-side JavaScript to get history data asynchronously
- More user friendly action views

1.9.28 0.1.2 (2013-08-6)

- Admin working
- Better publish statement
- Fix dependencies

1.9.29 0.1.1 (2013-08-6)

- Ported internals from django-stripe-payments
- Began writing the views
- Travis-CI
- All tests passing on Python 2.7 and 3.3
- All tests passing on Django 1.4 and 1.5
- Began model cleanup
- Better form
- Provide better response from management commands

1.9.30 0.1.0 (2013-08-5)

- First release on PyPI.

CHAPTER 2

Constraints

1. For stripe.com only
2. Only use or support well-maintained third-party libraries
3. For modern Python and Django

Index

T

U

V