Authorize Sauce Documentation

Release 0.4.1

Jeff Schenck

March 19, 2016

Contents

The secret sauce for accessing the Authorize.net API. The Authorize APIs for transactions, recurring payments, and saved payments are all different and awkward to use directly. Instead, you can use Authorize Sauce, which unifies all three Authorize.net APIs into one coherent Pythonic interface. Charge credit cards, easily!

```
>>> # Init the authorize client and a credit card
>>> from authorize import AuthorizeClient, CreditCard
>>> client = AuthorizeClient('285tUPuS', '58JKJ4T95uee75wd')
>>> cc = CreditCard('4111111111111111', '2018', '01', '911', 'Joe', 'Blow')
>>> card = client.card(cc)

>>> # Charge a card
>>> card.capture(100)
<AuthorizeTransaction 2171829470>

>>> # Save the card on Authorize servers for later
>>> saved_card = card.save()
>>> saved_card.uid
'7713982|6743206'

>>> # Use a saved card to auth a transaction, and settle later
>>> saved_card = client.saved_card('7713982|6743206')
>>> transaction = saved_card.auth(200)
>>> transaction.settle()
```

CHAPTER 1

Saucy Features

- Charge a credit card
- Authorize a credit card charge, and settle it or release it later
- Credit or refund to a card
- Save a credit card securely on Authorize.net's servers
- Use saved cards to charge, auth and credit
- Create recurring charges, with billing cycles, trial periods, etc.

Thanks to Chewse for supporting the development and open-sourcing of this library. Authorize Sauce is released under the MIT License.

CHAPTER 2

Contents

2.1 Installation

2.1.1 Install with pip

Simply use pip to install the authorize package

```
pip install authorizesauce
```

2.1.2 Install from source

Download or clone the source from Github and run setup.py install

```
git clone http://github.com/jeffschenck/authorizesauce.git
cd authorizesauce
python setup.py install
```

2.1.3 Requirements

Authorize Sauce has two external dependencies:

- six
- suds-jurko

If you want to build the docs or run the tests, there are additional dependencies, which are covered in the Development section.

2.2 Introduction

Here you'll find an easy introduction to charging credit cards with Authorize Sauce. We'll take you through the basics of charging, saving cards, and creating recurring payments.

For the full scoop on interacting with the Authorize Sauce client, see the Authorize interface documentation.

2.2.1 First, some terminology

The payments world uses some pretty wonky terms for working with credit cards, so let's define our terms right up front.

capture The credit card is first authorized for the given transaction amount, and if approved, is automatically submitted for settlement. This is very much like when a store clerk swipes your card at the register.

auth The credit card is temporarily authorized for a given amount without actually submitting it for settlement. This allows you to guarantee you'll be able to charge the card, but to hold off in case you later need to release the authorization or charge a lower amount. This transaction is not completed until you explicitly submit it for settlement.

settle A previous authorization transaction is submitted for settlement. This can be for any amount up to the original authorization amount.

void Cancel a previous authorization transaction.

credit Refunds part or all of a previously settled transaction. (Note that it must actually be settled, not just submitted for settlement. This can take up to 24 hours.)

2.2.2 Initialize the client

Whatever you plan to do, your first step will be to create the *AuthorizeClient* instance using your Authorize.net API login and transaction key:

```
>>> from authorize import AuthorizeClient
>>> client = AuthorizeClient('285tUPuS', '58JKJ4T95uee75wd')
```

2.2.3 Charge a credit card

Using the Authorize.net client, we can now create a *CreditCard* object and create a $100 charge with it. We'll also store a reference to the transaction just in case we need to refund it later:

```
>>> from authorize import CreditCard
>>> cc = CreditCard('4111111111111111', '2018', '01', '911', 'Joe', 'Blow')
>>> transaction = client.card(cc).capture(100)
>>> # Save the uid for this charge in case we need to refund it later
>>> transaction.uid
'2171830830'
```

Oh crap, someone wants a refund. That sucks for business, but at least it's not hard to do in Awesome Sauce:

```
>>> # Reference the transaction from earlier
>>> transaction = client.transaction('2171830830')
>>> # Refund the earlier transaction, passing in the last four digits of the card for verification
>>> transaction.credit('1111', 100)
<AuthorizeTransaction 2171830830>
```

2.2.4 Authorize a credit card

If you want to simply authorize a credit card for a certain amount, but don't want to actually settle that charge until later, we make that easy too! Let's start by authorizing a $100 payment:

```
>>> cc = CreditCard('4111111111111111', '2018', '01', '911', 'Joe', 'Blow')
>>> transaction = client.card(cc).auth(100)
>>> # Save the uid for this auth so we can settle it at a later date
>>> transaction.uid
'2171830878'
```

So let's say we've rendered services and we're ready to settle that $100 transaction from earlier. Easy:

```
>>> # Reference the transaction from earlier
>>> transaction = client.transaction('2171830878')
>>> transaction.settle()
<AuthorizeTransaction 2171830878>
```

But what if the total your customer owed came out to be less than that original $100 authorization? You can just as easily capture a smaller amount than the original authorization:

```
>>> # Reference the transaction from earlier
>>> transaction = client.transaction('2171830878')
>>> transaction.settle(50)
<AuthorizeTransaction 2171830878>
```

2.2.5 Save a credit card

Let's say you want to save a customer's credit card to make it easier for them to check out next time they're on your site:

```
>>> saved_card = client.card(cc).save()
>>> # Save the uid of the saved card so you can reference it later
>>> saved_card.uid
'7715743|6744936'
```

Now all you have to do is save that `uid` in your database, instead of storing sensitive credit card data, and you can charge the card again later.

```
>>> # Reference the saved card uid from earlier
>>> saved_card = client.saved_card('7715743|6744936')
>>> # Let's charge another $500 to this card for another purchase
>>> saved_card.capture(500)
<AuthorizeTransaction 2171830935>
```

If your user ever requests that you delete this card from its secure storage on Authorize.net's servers, we can do that too:

```
>>> saved_card = client.saved_card('7715743|6744936')
>>> saved_card.delete()
```

2.2.6 Create a recurring payment

Next you decide you want recurring revenue, so you're going to charge your users a monthly $20 subscription fee starting Dec 1, 2012. This is simple to set up:

```
>>> from datetime import date
>>> card = client.card(cc)
>>> card.recurring(20, date(2012, 12, 1), months=1)
<AuthorizeRecurring 1396734>
```

Again, if you want to update the recurring payment, this is easy to do. Let's say we need to increase the monthly rate to $25:

```
>>> # Reference the recurring uid from earlier
>>> recurring = client.recurring('1396734')
>>> recurring.update(amount=25)
```

And if the user cancels their service, we can end their recurring payment:

```
>>> recurring = client.recurring('1396734')
>>> recurring.delete()
```

There are many other available options when setting up recurring payments, such as trial periods and limited number of payments. For details, see the *AuthorizeCreditCard.recurring* method documentation.

2.3 Payment data

This module provides the data structures for describing credit cards and addresses for use in executing charges.

2.3.1 Credit card

class authorize.data.**CreditCard**(*card_number=None*, *exp_year=None*, *exp_month=None*, *cvv=None*, *first_name=None*, *last_name=None*)

Represents a credit card that can be charged.

Pass in the credit card number, expiration date, CVV code, and optionally a first name and last name. The card will be validated upon instatiation and will raise an *AuthorizeInvalidError* for invalid credit card numbers, past expiration dates, etc.

card_type

The credit card issuer, such as Visa or American Express, which is determined from the credit card number. Recognizes Visa, American Express, MasterCard, Discover, and Diners Club.

expiration

The credit card expiration date as a `datetime` object.

safe_number

The credit card number with all but the last four digits masked. This is useful for storing a representation of the card without keeping sensitive data.

validate()

Validates the credit card data and raises an *AuthorizeInvalidError* if anything doesn't check out. You shouldn't have to call this yourself.

2.3.2 Address

class authorize.data.**Address**(*street=None*, *city=None*, *state=None*, *zip_code=None*, *country='US'*)

Represents a billing address for a charge. Pass in the street, city, state and zip code, and optionally country for the address.

2.4 Authorize interface

This is your main interface to the Authorize.net API, where you feed in your credentials and then interact to create transactions and so forth. You will need to sign up for your own developer account and credentials at Authorize.net.

Warning: To use the saved card and recurring billing functionality, you must have either or both set up in your Authorize.net account. You must sign up your account for the CIM (Customer Information Manager) service and/or the ARB (Automated Recurring Billing) service, each of which may be an additional monthly charge. See the *Authorize.net documentation* for additional information.

2.4.1 Authorize client

class `authorize.client.`**`AuthorizeClient`**(*login_id*, *transaction_key*, *debug=True*, *test=False*)

Instantiate the client with your login ID and transaction key from Authorize.net.

The `debug` option determines whether to use debug mode for the APIs. This should be `True` in development and staging, and should be `False` in production when you want to actually process credit cards. You will need to pass in the appropriate login credentials depending on debug mode. The `test` option determines whether to run the standard API in test mode, which should generally be left `False`, even in development and staging environments.

`card`(*credit_card*, *address=None*, *email=None*)

To work with a credit card, pass in a *`CreditCard`* instance, and optionally an *`Address`* instance. This will return an *`AuthorizeCreditCard`* instance you can then use to execute transactions. `email` is only required for those using European payment processors.

`recurring`(*uid*)

To update or cancel an existing recurring payment, pass in the `uid` of the recurring payment as a string. This will return an *`AuthorizeRecurring`* instance you can then use to udpate or cancel the payments.

`saved_card`(*uid*)

To create a new transaction from a saved card, pass in the `uid` of the saved card as a string. This will return an *`AuthorizeSavedCard`* instance you can then use to auth, capture, or create a credit.

`transaction`(*uid*)

To perform an action on a previous transaction, pass in the `uid` of that transaction as a string. This will return an *`AuthorizeTransaction`* instance you can then use to settle, credit or void that transaction.

2.4.2 Credit card

class `authorize.client.`**`AuthorizeCreditCard`**(*client*, *credit_card*, *address=None*, *email=None*)

This is the interface for working with a credit card. You use this to authorize or charge a credit card, as well as saving the credit card and creating recurring payments.

Any operation performed on this instance returns another instance you can work with, such as a transaction, saved card, or recurring payment.

`auth`(*amount*)

Authorize a transaction against this card for the specified amount. This verifies the amount is available on the card and reserves it. Returns an *`AuthorizeTransaction`* instance representing the transaction.

`capture`(*amount*)

Capture a transaction immediately on this card for the specified amount. Returns an *`AuthorizeTransaction`* instance representing the transaction.

recurring(*amount*, *start*, *days=None*, *months=None*, *occurrences=None*, *trial_amount=None*, *trial_occurrences=None*)

Creates a recurring payment with this credit card. Pass in the following arguments to set it up:

amount The amount to charge at each interval.

start The `date` or `datetime` at which to begin the recurring charges.

days The number of days in the billing cycle. You must provide either the `days` argument or the `months` argument.

months The number of months in the billing cycle. You must provide either the `days` argument or the `months` argument.

occurrences ***(optional)*** The number of times the card should be billed before stopping. If not specified, it will continue indefinitely.

trial_amount ***(optional)*** If you want to charge a lower amount for an introductory period, specify the amount.

trial_occurrences ***(optional)*** If you want to charge a lower amount for an introductory period, specify the number of occurrences that period should last.

Returns an *AuthorizeRecurring* instance that you can save, update or delete.

save()

Saves the credit card on Authorize.net's servers so you can create transactions at a later date. Returns an *AuthorizeSavedCard* instance that you can save or use.

2.4.3 Transaction

class `authorize.client`.**AuthorizeTransaction**(*client*, *uid*)

This is the interface for working with a previous transaction. It is returned by many other operations, or you can save the transaction's `uid` and reinstantiate it later.

You can then use this transaction to settle a previous authorization, credit back a previous transaction, or void a previous authorization. Any such operation returns another transaction instance you can work with.

Additionally, if you need to access the full raw result of the transaction it is stored in the `full_response` attribute on the class.

credit(*card_number*, *amount*)

Creates a credit (refund) back on the original transaction. The `card_number` should be the last four digits of the credit card and the `amount` is the amount to credit the card. Returns an *AuthorizeTransaction* instance representing the credit transaction.

Credit transactions are bound by a number of restrictions:

- The original transaction must be an existing, settled charge. (Note that this is different than merely calling the *AuthorizeTransaction.settle* method, which submits a payment for settlement. In production, Authorize.net actually settles charges once daily. Until a charge is settled, you should use *AuthorizeTransaction.void* instead.)
- The amount of the credit (as well as the sum of all credits against this original transaction) must be less than or equal to the original amount charged.
- The credit transaction must be submitted within 120 days of the date the original transaction was settled.

settle(*amount=None*)

Settles this transaction if it is a previous authorization. If no `amount` is specified, the full amount will be settled; if a lower `amount` is provided, the lower amount will be settled; if a higher `amount` is given,

it will result in an error. Returns an *AuthorizeTransaction* instance representing the settlement transaction.

void()
Voids a previous authorization that has not yet been settled. Returns an *AuthorizeTransaction* instance representing the void transaction.

2.4.4 Saved card

class authorize.client.**AuthorizeSavedCard**(*client*, *uid*)
This is the interface for working with a saved credit card. It is returned by the *AuthorizeCreditCard.save* method, or you can save a saved card's `uid` and reinstantiate it later.

You can then use this saved card to create new authorizations, captures, and credits. Or you can delete this card from the Authorize.net database. The first three operations will all return a transaction instance to work with.

You can also retrieve payment information with the `AuthorizeSavedCard.get_payment_info <authorize.client.AuthorizeSavedCard.get_payment_info()` method.

You can update this information by setting it and running the `AuthorizeSavedCard.update` method.

auth(*amount*, *cvv=None*)
Authorize a transaction against this card for the specified amount. This verifies the amount is available on the card and reserves it. Returns an *AuthorizeTransaction* instance representing the transaction.

capture(*amount*, *cvv=None*)
Capture a transaction immediately on this card for the specified amount. Returns an *AuthorizeTransaction* instance representing the transaction.

delete()
Removes this saved card from the Authorize.net database.

2.4.5 Recurring charge

class authorize.client.**AuthorizeRecurring**(*client*, *uid*)
This is the interface for working with a recurring charge. It is returned by the *AuthorizeCreditCard.recurring* method, or you can save a recurring payment's `uid` and reinstantiate it later.

The recurring payment will continue charging automatically, but if you want to make changes to an existing recurring payment or to cancel a recurring payment, this provides the interface.

delete()
Cancels any future charges from this recurring payment.

update(*amount=None*, *start=None*, *occurrences=None*, *trial_amount=None*, *trial_occurrences=None*)
Updates the amount or status of the recurring payment. You may provide any or all fields and they will be updated appropriately, so long as none conflict. Fields work as described under the

amount ***(optional)*** The amount to charge at each interval. Will only be applied to future charges.

start ***(optional)*** The `date` or `datetime` at which to begin the recurring charges. You may only specify this option if the recurring charge has not yet begun.

occurrences ***(optional)*** The number of times the card should be billed before stopping. If not specified, it will continue indefinitely.

trial_amount *(optional)* If you want to charge a lower amount for an introductory period, specify the amount. You may specify this option only if there have not yet been any non-trial payments.

trial_occurrences *(optional)* If you want to charge a lower amount for an introductory period, specify the number of occurrences that period should last. You may specify this option only if there have not yet been any non-trial payments.

2.5 Exceptions

class authorize.exceptions.**AuthorizeError**
Base class for all errors.

class authorize.exceptions.**AuthorizeConnectionError**
Error communicating with the Authorize.net API.

class authorize.exceptions.**AuthorizeResponseError**
Error response code returned from API.

class authorize.exceptions.**AuthorizeInvalidError**
Invalid information provided.

2.6 Development

All assistance is appreciated! New features, documentation fixes, bug reports, bug fixes, and more are graciously accepted.

2.6.1 Getting started

To get set up, fork the project on our Github page. You can then install from source by following the instructions in Installation. There are a few additional dependencies for compiling the docs and running the tests:

- mock
- unittest2 (if on Python < 2.7)
- sphinx (for docs only)

You can install all dependencies using pip from the `requirements.txt` file:

```
pip install -r requirements.txt
```

2.6.2 Running the tests

Once you're all installed up, ensure that the tests pass by running them. You can run the local unit tests with the following command:

```
./tests/run_tests.py
```

However, the above command skips some integration tests that actually hit the remote Authorize.net test server. This is done so the main test suite can be run quickly and without an internet connection. However, you should occasionally run the full test suite, which can be done by setting an environment variable:

```
AUTHORIZE_LIVE_TESTS=1 ./tests/run_tests.py
```

2.6.3 Testing in all supported Python versions

Since Authorize Sauce supports multiple Python versions, running the tests in your currently installed Python version may not be enough. You can use tox to automate running the tests in all supported versions of Python.

First, install tox, if you haven't already:

```
pip install tox
# or easy_install tox
```

Running it is simple:

```
tox
```

If you want to only test against a subset of Python versions, you can do so:

```
tox -e py27,py34
```

As above, you can set an environment variable to run the full test suite in every supported Python version:

```
AUTHORIZE_LIVE_TESTS=1 tox
```

Note: tox requires that the tested Python versions are already present on your system. If you need to install one or more versions of Python, pyenv or (for Ubuntu) the deadsnakes PPA can be helpful.

2.6.4 Authorize.net documentation

The Authorize.net documentation somehow manages to be both overly verbose and fairly uninformative. That said, you can find it here:

- Developer site
- Advanced Integration Method
- Customer Information Manager
- Automated Recurring Billing

2.6.5 Submitting bugs and patches

If you have a bug to report, please do so on our Github issues page. If you've got a fork with a new feature or a bug fix with tests, please send us a pull request.

a

A

C

D

E

R

S

T

U

V