# pystrike Documentation

**Joseph Schilz**

**Nov 29, 2018**

# Contents:

Python wrapper for Acinq's Strike lightning network payment service.

The lightning network allows near-fee, near-instant transactions atop the Bitcoin chain layer. Acinq operates the Strike service, which allows you to create lightning invoices, receive lightning payments into your Strike account, and then receive consolidated payouts on-chain. This Python library allows you to invoice customers and check the payment-status of those invoices in just a few lines of code.

This library does not require any third-party dependencies.

# CHAPTER 1

# Example

Initialize the `Charge` class:

```
from pystrike.charge import make_charge_class

Charge = make_charge_class(
    api_key="YOURSTRIKETESTNETAPIKEY",
    api_host="api.dev.strike.acinq.co",
    api_base="/api/v1/",
)
```

Create a new `charge`:

```
charge = Charge(
        currency=Charge.CURRENCY_BTC,
        amount=4200,                    # Amount in Satoshi
        description="services rendered",
    )
```

Retrieve a payment request:

```
payment_request = charge.payment_request

# Now `payment_request` might be something like "lnbtb420u1pfoobarbaz..."
```

At this point, you would present the `payment_request` to your customer. You can call `charge.update()` to poll the Strike server for the current status of the charge, and then retrieve whether or not the charge has been paid from the `charge.paid` attribute.

For example, suppose that `charge.payment_request` has not yet been paid and then we run the following code:

```
charge.update()  # Reaches out the the Acinq server to retrieve the
                 # status of the charge

paid = charge.paid
# Because the payment request has not yet been paid, charge.paid is False
```

Then suppose that the client pays the `charge.payment_request` and then we run the following code:

```
charge.update()
paid = charge.paid
# Because the client paid the request before we called `update`, charge.paid
# evaluates to True.
```

Acinq's Strike service also offers a web hook/callback service, which is a better way to update your charges than frequent polling if you are running a web service.

The example above uses Strike's testnet web service at `api.dev.strike.acinq.co`. When you're ready to issue mainnet lightning invoices, you'll need to use your Strike mainnet API key and make your requests to host `api.strike.acinq.co`.

CHAPTER 2

# Use

## 2.1 Install pystrike

```
$ pip install pystrike
```

## 2.2 Creating an API Key

Begin by creating an account on Acinq's Strike lightning network payment service. Note that there is also a testnet version of the service that you may wish to use for your initial development. The two versions of this service are distinct, with separate accounts, separate API keys, and separate API hosts.

Once you have created an account and logged into the dashboard, you can retrieve an API key from your dashboard settings. You will need this key to configure your connection to Strike.

## 2.3 Configuring a Charge Class

You'll begin by creating a Charge class from the provided `make_charge_class` function.

```
from pystrike.charge import make_charge_class

Charge = make_charge_class(
    api_key="YOURSTRIKETESTNETAPIKEY",
    api_host="api.dev.strike.acinq.co",
    api_base="/api/v1/",
)
```

The host will probably be one of:

- api.strike.acinq.co: the mainnet version of Strike
- api.dev.strike.acinq.co: the testnet version of Strike

If you're pointing your charge class to the mainnet version then be sure to use the API key from your mainnet Strike dashboard.

## 2.4 Creating a Charge

You can create a new charge with the following code:

```
charge = Charge(
        currency=Charge.CURRENCY_BTC,
        amount=4200,                    # Amount in Satoshi
        description="services rendered",
    )
```

This initialization will automatically reach out to the Strike web service and create a new charge on their servers. Once this call has returned, you can immediately access the details of that charge through `charge.id`, `charge.payment_request`, and so on.

At this point, you might present the `charge.payment_request` to your customer for payment.

## 2.5 Retrieving a Charge

Rather than creating a new charge, if you know the Strike id of an existing charge you can retrieve it with the following code:

```
charge = Charge.from_charge_id('ch_LWafoobarbazjFFv8eurFJkerhgDA')
```

## 2.6 Updating a Charge

You can poll the Strike server to update your local charge object:

```
charge.update()
```

This command reaches out to the Strike server and updates the attributes of the charge. For example, if you are waiting on payment for a charge, you might run `charge.update()` to retrieve the status of the charge from the Strike server and then access `charge.paid` to see if a payment has been recorded for the charge on the Strike server.

If you're developing a web application, you could use web hooks instead of polling the server. See Strike's documentation on web hooks for more information.

CHAPTER 3

# Testing

Running the library tests requires two environment variables:

- `STRIKE_TESTNET_API_KEY`: Your API key for the `api.dev.strike.acinq.co` web service.
- `RETRIEVE_CHARGE_ID`: The Strike id of a charge in your `api.dev.strike.acinq.co`. For example: `ch_LWafoobarbazjFFv8eufoobarbaz`

CHAPTER 4

# API Guide

## 4.1 pystrike.charge.Charge

**class** pystrike.charge.**Charge**(*amount*, *currency*, *description=''*, *customer_id=''*, *create=True*)

The Charge class is your interface to the Strike web service. Use it to create, retrieve, and update lighting network charges.

Each instance is a lazy mirror, reflecting a single charge on the Strike servers. The instance is lazy in that it will communicate with Strike implicitly, but only as needed.

When you initialize a charge with an amount and description, the instance does not create an instance on Strike until the moment that you request an attribute such as *payment_request*. If you request the charge's *paid* attribute, then the charge will update itself from the Strike server if it has not yet seen its payment clear; but if *paid* is already set to *True* then the charge will simply report *True* without reaching out to the server.

**Variables**

- **amount** – The amount of the invoice, in self.currency.
- **currency** – The currency of the request.
- **description** – Narrative description of the invoice.
- **customer_id** – An optional customer identifier.
- **id** – The id of the charge on Strike's server.
- **amount_satoshi** – The amount of the request, in satoshi.
- **payment_request** – The payment request string for the charge.
- **payment_hash** – The hash of the payment for this charge.
- **paid** – Whether the request has been satisfied.
- **created** – When the charge was created, in epoch time.
- **updated** – When the charge was updated, in epoch time.

**__init__**(*amount*, *currency*, *description=''*, *customer_id=''*, *create=True*)
Initialize an instance of *Charge*. See the Strike API documentation for details on each of the arguments.

**Args:**

- amount (int): The amount of the charge, in Satoshi.
- currenency (str): Must be *Charge.CURRENCY_BTC*.

**Kwargs:**

- description (str): Optional invoice description.
- customer_id (str): Optional customer identifier.
- **create (bool): Whether to automatically create a** corresponding charge on the Strike service.

**__getattribute__**()
Return getattr(self, name).

**api_base**
Concrete subclasses must define an api_base.

**api_host**
Concrete subclasses must define an api_host.

**api_key**
Concrete subclasses must define an api_key.

**classmethod from_charge_id**(*charge_id*)
Class method to create and an instance of *Charge* and fill it from the Strike server.

**Args:**

- charge_id (str): The id of a charge on Strike's server.

**Returns:**

- An instance of *Charge*, filled from the attributes of the charge with the given *charge_id*.

**update**()
Update the charge from the server.

If this charge has an *id*, then the method will _retrieve_ the charge from the server. If this charge does not have an *id*, then this method will _create_ the charge on the server and then fill the local charge from the attributes created and returned by the Strike server.

## 4.2 pystrike.charge.make_charge_class

pystrike.charge.**make_charge_class**(*api_key*, *api_host*, *api_base*)
Generates a Charge class with the given parameters

**Args:**

- api_key (str): An API key associated with your Strike account.
- **api_host (str): The host name of the Strike server you'd like** to connect to. Probably one of: - "api.strike.acinq.co" - "api.dev.strike.acinq.co"
- **api_base (str): The base path of the Strike API on the host** server. Probably: "/api/v1/"

**Returns:** A parameterized Charge class object.

# Index

## Symbols



## A



## C



## F



## M



## U