

---

# **CryptoToken Converter Documentation**

**Privex Inc., Chris (Someguy123)**

**Aug 25, 2019**



---

## Contents:

---

<b>1</b>	<b>Installation and configuration</b>	<b>3</b>
1.1	Requirements and Dependencies . . . . .	3
1.2	Install Core Dependencies . . . . .	4
1.3	Create Database and DB user . . . . .	4
1.4	Download and install the project . . . . .	5
1.5	Beem Wallet (if using Steem) . . . . .	6
1.6	Basic Configuration . . . . .	6
1.7	Final Setup . . . . .	7
1.8	Transaction Scanning and Conversion . . . . .	8
<b>2</b>	<b>Running in Production</b>	<b>9</b>
<b>3</b>	<b>Python Code Documentation</b>	<b>11</b>
3.1	<i>steemengine</i> module . . . . .	11
3.1.1	steemengine package . . . . .	11
3.1.1.1	Subpackages . . . . .	11
3.1.1.1.1	steemengine.settings (Django Settings) . . . . .	11
3.1.1.1.1.1	Submodules . . . . .	11
3.1.1.1.1.2	steemengine.settings.core module . . . . .	11
3.1.1.1.1.3	steemengine.settings.custom module . . . . .	12
3.1.1.1.1.4	steemengine.settings.log module . . . . .	13
3.1.1.1.1.5	Module contents . . . . .	14
3.1.1.2	Submodules . . . . .	14
3.1.1.3	steemengine.helpers module . . . . .	14
3.1.1.4	steemengine.urls module . . . . .	16
3.1.1.5	steemengine.wsgi module . . . . .	16
3.1.1.6	Module contents . . . . .	17
3.2	<i>payments</i> module . . . . .	17
3.2.1	Coin Handlers . . . . .	17
3.2.1.1	Subpackages . . . . .	17
3.2.1.1.1	Bitcoind Coin Handler . . . . .	17
3.2.1.1.1.1	Module contents . . . . .	17
3.2.1.1.1.2	Submodules . . . . .	18
3.2.1.1.1.3	BitcoinLoader module . . . . .	18
3.2.1.1.1.4	BitcoinManager module . . . . .	20
3.2.1.1.1.5	BitcoinMixin module . . . . .	22
3.2.1.1.2	SteemEngine Coin Handler . . . . .	23

3.2.1.1.2.1	Module contents . . . . .	23
3.2.1.1.2.2	Submodules . . . . .	23
3.2.1.1.2.3	SteemEngineLoader module . . . . .	23
3.2.1.1.2.4	SteemEngineManager module . . . . .	24
3.2.1.1.3	Steem Coin Handler . . . . .	28
3.2.1.1.3.1	Module contents . . . . .	28
3.2.1.1.3.2	Submodules . . . . .	29
3.2.1.1.3.3	SteemLoader module . . . . .	29
3.2.1.1.3.4	SteemManager module . . . . .	31
3.2.1.1.4	Coin Handler Base Classes . . . . .	34
3.2.1.1.4.1	Submodules . . . . .	34
3.2.1.1.4.2	BaseLoader . . . . .	34
3.2.1.1.4.3	BaseManager . . . . .	35
3.2.1.1.4.4	BatchLoader . . . . .	39
3.2.1.1.4.5	Base Decorators . . . . .	42
3.2.1.1.4.6	Base Exceptions . . . . .	43
3.2.1.1.4.7	Module contents . . . . .	43
3.2.1.2	Module contents . . . . .	43
3.2.2	payments package . . . . .	46
3.2.2.1	Subpackages . . . . .	46
3.2.2.2	Submodules . . . . .	46
3.2.2.3	payments.admin module . . . . .	46
3.2.2.4	payments.apps module . . . . .	48
3.2.2.5	payments.models module . . . . .	48
3.2.2.6	payments.serializers module . . . . .	59
3.2.2.7	payments.tests module . . . . .	60
3.2.2.8	payments.views module . . . . .	60
3.2.2.9	Module contents . . . . .	61
<b>4</b>	<b>REST API Documentation</b>	<b>63</b>
4.1	Endpoints . . . . .	63
4.2	/api/convert/ . . . . .	63
4.3	/api/deposits/ . . . . .	65
4.4	/api/conversions/ . . . . .	67
4.5	/api/pairs/ . . . . .	69
<b>5</b>	<b>Indices and tables</b>	<b>73</b>
	<b>Python Module Index</b>	<b>75</b>
	<b>Index</b>	<b>77</b>

This documentation is for CryptoToken Converter, an open source project developed by [Privex Inc.](#) for simple, anonymous conversion between two cryptocurrencies / tokens.

You can find the full source code for the project on our [Github](#)

It allows for both uni-directional and bi-directional conversions between user specified coin pairs, whether that's BTC->LTC, or LTC->LTCP (Litecoin to Pegged Litecoin token).

Using CryptoToken Converter, you can easily operate services such as crypto <-> token gateways, as well as token <-> token and crypto <-> crypto.

Out of the box, CryptoToken Converter comes with two *Coin Handlers*:

- ***Bitcoin* Coin Handler** Handles deposits/sending for coins which have a fork of *bitcoin* without dramatic JSONRPC API changes (e.g. Litecoin, Dogecoin).
- ***SteemEngine* Coin Handler** Handles deposits/issuing/sending for tokens that exist on the [Steem Engine](#) platform - a side-chain of the [Steem](#) blockchain.

Every “coin pair” has an exchange rate set in the database, which can be either statically set for pegged tokens, or dynamically updated for conversions between two different cryptos/tokens.



# CHAPTER 1

---

## Installation and configuration

---

**Attention:** This guide is aimed at Ubuntu Bionic 18.04 - if you're not running Ubuntu 18.04, some parts of the guide may not apply to you, or simply won't work.

---

**Tip:** If you don't have any machines running Ubuntu 18.04, you can grab a [dedicated or virtual server](#) pre-installed with it from [Privex](#) - we're the ones who wrote this software! :)

---

## 1.1 Requirements and Dependencies

### Core Dependencies

- Python 3.7+ (may or may not work on older versions)
- PostgreSQL or MySQL for the database
- Nginx for the production web server
- Linux or macOS (OSX) is recommended (may work on Windows, however we refuse to actively support it)

### Additional Requirements

- If you plan to use the *Bitcoin* [Coin Handler](#) you'll need one or more coin daemons such as `bitcoind`, `litecoind` or `dogecoind` running in server mode, with an `rpcuser` and `rpcpassword` configured.
- If you plan to use the *Steem* [Engine Coin Handler](#) you'll need a [Steem account](#) - for best operation it's recommended that you use [Steem Engine](#) tokens that you've created (can issue them), and you must have the **active private key** of the token owner account.

### Knowledge

- You should have basic knowledge of navigating a Linux/Unix system, including running basic commands
- It may help if you have at least a basic understanding of the Python programming language

- If you plan to contribute to the project, or make modifications, you should read the documentation for the [Django Framework](#), and the third-party add-on [Django REST Framework](#)

## 1.2 Install Core Dependencies

For this guide, we'll be using PostgreSQL, but you're free to use MySQL if you're more comfortable with it.

Using your system package manager, install Python 3.7, Postgres server, nginx, git, along with some various important libraries needed for our Python packages.

```
sudo apt update
# Install Python 3.7, Nginx, and Git
sudo apt install -y python3.7 python3.7-dev python3.7-venv nginx git

# Install libssl-dev for the OpenSSL headers (required for the Beem python library)
# and build-essential - various tools required for building and compiling the python_
↳dependencies
sudo apt install -y build-essential libssl-dev

# The `postgresql` package will install the latest Postgres client and server, we_
↳also want libpq-dev,
# which is the postgres client dev headers, sometimes needed for Python postgres_
↳libraries
sudo apt install -y postgresql libpq-dev

# Install MariaDB (cross-compatible with MySQL) and the development headers to avoid_
↳issues with the Python
# MySQL library
sudo apt install -y mariadb-server libmariadbclient-dev libmariadb-dev
```

---

**Tip:** The below step for setting your default python3 is optional, but it may help prevent issues when python files refer to python3 and not python3.7

---

To avoid the issue of python3 referring to an older version of Python 3, you should run the following commands to set up Python 3.7 as the default. On Ubuntu 18.04, Python 3.6 is the default used for python3.

```
# Make sure both Python 3.6 (Ubuntu 18.04 default), and 3.7 are registered with_
↳update-alternatives
sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.6 1
sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.7 2
# Set `python3.7` as the default version to use when `python3` is ran
sudo update-alternatives --set python3 /usr/bin/python3.7
```

To check if the above worked, you should see 3.7.x when running `python3 -V` like below:

```
user@host ~ $ python3 -V
Python 3.7.1
```

## 1.3 Create Database and DB user

For Postgres, this is very easy.

Simply run the below commands to create a user, a database, and make the user the owner of the DB.



```
# Log in as the postgres user
root@host # su - postgres

# Create a user, you'll be prompted for the password
# S = not a superuser, D = cannot create databases, R = cannot create roles
# l = can login, P = prompt for user's new password
$ createuser -SDRl -P steemengine
    Enter password for new role:
    Enter it again:

# Create the database steemengine_pay with the new user as the owner

$ createdb -O steemengine steemengine_pay

# If you've already created the DB, use psql to manually grant permissions to the user

$ psql
psql (10.6 (Ubuntu 10.6-0ubuntu0.18.04.1))
Type "help" for help.

postgres=# GRANT ALL ON DATABASE steemengine TO steemengine_pay;
```

The above commands create a postgres user called steemengine and a database called steemengine\_pay .  
Feel free to adjust the username and database name to your liking.

## 1.4 Download and install the project

**Tip:** If you're running this in production, for security you should create a limited account, and install the project using that account.

Clone the repo, and enter the directory.

```
git clone https://github.com/privex/cryptotoken-converter
cd cryptotoken-converter
```

Create and activate a **python virtual environment** to avoid conflicts with any packages installed system-wide, or any upgrades to the python version.

```
# Create the virtual environment in the folder `venv`
python3.7 -m venv venv
# Activate the virtual environment.
source venv/bin/activate
```

You must make sure to activate the virtualenv before you run any python files, or install any python packages.

While the virtualenv is activated, you'll see the text (venv) on the side of your shell, like so:

```
(venv) user@host ~/cryptotoken-converter $
```

Now that the virtualenv is created and activated, we can install the python packages required to run this project.

```
# pip3 is the package manager for Python 3, this command will install the packages_
↳ listed in `requirements.txt`
pip3 install -r requirements.txt
```

## 1.5 Beem Wallet (if using Steem)

If you're using a coin handler that uses the **Steem network**, such as *SteemEngine Coin Handler*, then you must create a Beem wallet, and add the **active private key** for each Steem account you intend to send/issue from.

```
# Create a new Beem wallet, make sure to remember your wallet password, you'll need_
↳ it later.
beempy createwallet
# Import the Private Active Key for each Steem account you plan to send/issue from.
beempy addkey
```

## 1.6 Basic Configuration

The first step of configuration is creating a `.env` file, this will contain various configuration details needed to run the project.

```
# Creates a file called `.env` if it doesn't already exist
touch .env
# Ensures that `.env` can only be read/written to by your user.
chmod 700 .env
```

Open up `.env` in your favourite text editor (such as `vim` or `nano`).

Paste the following example config:

```
DB_USER=steemengine_pay
DB_PASS=MySuperSecretPassword
DB_NAME=steemengine
DEBUG=false
SECRET_KEY=VeryLongRandomStringUsedToProtectYourUserSessions
UNLOCK=
```

Some of the above options can simply be left out if they're just the default, but it's best to specify them anyway, to avoid the application breaking due to changes to the default values.

Now we'll explain what the above options do, as well as some extras.

### Basic Config

#### SECRET\_KEY - Required

A long (recommended 40+ chars) random string of uppercase letters, lowercase letters, and numbers. It's used for various Django functionality, including encryption of your user sessions/cookies.

#### DEBUG - Optional

If set to `True` Django will output detailed error pages, automatically re-load the app when python files are modified, among other helpful development features. If not specified, it defaults to `False`.

**This should always be set to `FALSE` in production, otherwise the error pages WILL leak a lot of information, including sensitive details such as passwords or API keys.**

**EX\_FEE - Optional**

This option sets the exchange fee, as a percentage. For example *1* would mean a 1% fee is taken from each exchange from crypto->token and token->crypto.

You may also use decimal numbers, such as *0.5* for 0.5%, or to disable exchange fees, simply set it to *0* or remove the line entirely, **as the default is no fee**.

**COIN\_HANDLERS - Optional.**

If you're using any third party *Coin Handlers* or you want to disable some of the default ones, this is a list of comma separated Coin Handler folder names.

**Default:** SteemEngine,Bitcoin

**Steem Configuration**

If you plan to use *SteemEngine Coin Handler* then you may want to configure these as needed.

**STEEM\_RPC\_NODES - Optional**

If you want to override the Steem RPC node(s) used for functions such as signing the custom\_json transactions from the token issuing account, you can specify them as a comma separated list.

They will be used in the order they are specified.

**Default:** Automatically use best node determined by Beem

**Example:** STEEM\_RPC\_NODES=https://steemd.privex.io,https://api.steemit.com

**UNLOCK - Required if using Steem**

The wallet password for Beem. This must be specified to allow Steem transactions to be automatically signed. See the section *Beem Wallet (if using Steem)* to create a wallet.

**Database Configuration**

- DB\_BACKEND - What type of DB are you using? mysql or postgresql Default: postgresql
- DB\_HOST - What hostname/ip is the DB on? Default: localhost
- DB\_NAME - What is the name of the database to use? Default: steemengine\_pay
- DB\_USER - What username to connect with? Default: steemengine
- DB\_PASS - What password to connect with? Default: no password

## 1.7 Final Setup

The app is almost ready to go! Just a few last things.

To create the database structure (tables, relations etc.), you'll need to run the Django migrations

```
./manage.py migrate
```

You'll also want to create an admin account (superuser)

```
./manage.py createsuperuser
```

Now, start the Django server

```
./manage.py runserver
```

You should now be able to go to <http://127.0.0.1:8000/admin/> in your browser and access the Django admin.

Login using the superuser account you've created.

Using the admin panel, create at least two Coin's (*payments.models.Coin*), and at least one Coin Pair (*payments.models.CoinPair*).

Make sure to set each Coin's "Coin Type" correctly, so that Coin Handlers will detect them (use the types "SteemEngine Token", and "Bitcoin Compatible"). You may have to refresh the "Add Coin" page if some of the types don't show up.

After adding the coins, you should now be able to open one of the API pages in your browser, such as this one: <http://127.0.0.1:8000/api/coins/>

If you can see your added coins on that page, everything should be working! :)

Now try making some conversions using the API: [REST API Documentation](#)

## 1.8 Transaction Scanning and Conversion

To handle incoming deposits, and converting deposits into their destination coin, there are two management commands to run.

```
./manage.py load_txs
```

The command **load\_txs** imports incoming transactions into the Deposits table for any Coin that has a properly configured Coin Handler (*Coin Handlers*).

```
./manage.py convert_coins
```

The command **convert\_coins** scans each deposit in the Deposit table to check if it's valid, and which Coin it should be converted to.

Each valid deposit will then be converted into its destination coin, and the deposit will be marked as *conv* (Successfully Converted).

If you're running with *DEBUG* set to true, you'll see a detailed log of what it's doing, so you can diagnose any problems with your coin configuration and fix it.

When running in production, you would normally have these running on a **cron** - a scheduled task.

To find out how to run this in production, please read [Running in Production](#)

## CHAPTER 2

---

### Running in Production

---



## 3.1 *steemengine* module

### 3.1.1 steemengine package

#### 3.1.1.1 Subpackages

##### 3.1.1.1.1 steemengine.settings (Django Settings)

###### 3.1.1.1.1.1 Submodules

###### 3.1.1.1.1.2 steemengine.settings.core module

This file contains the core settings of the application. Settings specified within this file are used directly by the Django framework, or a third-party extension / application for Django.

#### User specifiable environment variables:

##### Basic Config

- `DEBUG` - If set to true, enable debugging features, such as **extremely verbose error pages** and automatic code reloading on edit. **DO NOT RUN WITH DEBUG IN PRODUCTION, IT IS NOT SAFE.**  
**Default:** `False`
- `SECRET_KEY` - **MANDATORY** - A long random string used to encrypt user sessions, among other security features.
- `CORS_ORIGIN_ALLOW_ALL` - If True, allow all cross-origin requests (disable whitelist). **Default:** `True`
- `CORS_ORIGIN_WHITELIST` - Comma separated list of domains and subdomains to allow CORS for. Adding a domain does not automatically include it's subdomains. All subdomains must be added manually. **Default:** Blank

- `ALLOWED_HOSTS` - Comma separated list of the domains you intend to run this on. For security (e.g. preventing cookie theft), Django requires that you specify each hostname that this application should be accessible from. **Default:** `127.0.0.1, localhost` (these are also auto added if `DEBUG` is `True`).

### Database Settings

- `DB_BACKEND` - What type of DB are you using? `mysql` or `postgresql` **Default:** `postgresql`
- `DB_HOST` - What hostname/ip is the DB on? **Default:** `localhost`
- `DB_NAME` - What is the name of the database to use? **Default:** `steemengine_pay`
- `DB_USER` - What username to connect with? **Default:** `steemengine`
- `DB_PASS` - What password to connect with? **Default:** no password

For more information on this file, see <https://docs.djangoproject.com/en/2.1/topics/settings/>

For the full list of settings and their values, see <https://docs.djangoproject.com/en/2.1/ref/settings/>

Copyright:

```
+=====+
|                © 2019 Privex Inc.                |
|                https://www.privex.io              |
+=====+
|
|      CryptoToken Converter                        |
|
|      Core Developer(s):                          |
|
|      (+)  Chris (@someguy123) [Privex]            |
|
+=====+
```

### 3.1.1.1.3 steemengine.settings.custom module

This file contains settings that are specific to CryptoToken Converter, and do not affect the core Django framework.

User specifiable environment variables:

- `STEEM_RPC_NODES` - Comma-separated list of one/more Steem RPC nodes. If not set, will use the default beem nodes.
- `BITSHARES_RPC_NODE` - Node to use to connect to Bitshares network if Bitshares coin handler is enabled. If not set, will default to `wss://eu.nodes.bitshares.ws`
- `EX_FEE` - Conversion fee taken by us, in percentage (i.e. "1" = 1%) **Default:** 0 (no fee)
- `COIN_HANDLERS` - A comma separated list of Coin Handler modules to load. **Default:** `SteemEngine,Bitcoin`
- `COIN_HANDLERS_BASE` - If your coin handlers are not located in `payments.coin_handlers` then you may change this to point to the base module where your coin handlers are located.
- `LOWFUNDS_NOTIFY` - If you're using the low wallet balance notifications, you can change how often it re-notifies the admin emails `ADMINS` if the balance is still too low to fulfill a conversion. (in hours). **Default:** 12 (hours)

Copyright:



```

+=====+
|                © 2019 Privex Inc.                |
|                https://www.privex.io              |
+=====+
|
|                CryptoToken Converter              |
|
|                Core Developer(s) :                |
|
|                (+)  Chris (@someguy123) [Privex]   |
|
+=====+

```

#### 3.1.1.1.1.4 steemengine.settings.log module

Logging configuration for CryptoToken Converter.

Valid environment log levels (from least to most severe) are:

```

DEBUG, INFO, WARNING
ERROR, FATAL, CRITICAL

```

User customisable environment variables are:

- `CONSOLE_LOG_LEVEL` - Messages equal to and above this level will be logged to the console (i.e. the output of `manage.py` commands such as `runserver` or `load_txs`) **Default:** INFO in production, DEBUG when DEBUG setting is true
- `DBGFILE_LEVEL` - Messages equal to and above this level will be logged to the `debug.log` files. **Default:** INFO in production, DEBUG when DEBUG setting is true.
- `ERRFILE_LEVEL` - Same as `DBGFILE_LEVEL` but for `error.log` - **Default:** WARNING
- `LOGGER_NAMES` - A comma separated list of logger instance names to apply the default logging settings onto. **Default:** `privex` (Use same logging for Privex's python packages)
- `BASE_LOGGER_NAME` - The logger instance name to use for the main logger. If this is not specified, or is blank, then the logging API "RootLogger" will be used, which may automatically configure logging for various packages.
- `BASE_LOG_FOLDER` - A relative path from the root of the project (folder with `manage.py`) to the folder where log files should be stored. **Default:** `logs`
- `BASE_WEB_LOGS` - Relative path from `BASE_LOG_FOLDER` where logs from the web app should be stored. **Default:** `web`
- `BASE_CRON_LOGS` - Relative path from `BASE_LOG_FOLDER` where logs from scheduled commands (`load_txs` etc.) should be stored. **Default:** `crons`

`steemengine.settings.log.config_logger` (\*logger\_names, log\_dir='/home/docs/checkouts/readthedocs.org/user\_builds/converter/checkouts/stable/logs')

Used to allow isolated parts of this project to easily change the log output folder, e.g. allow Django management commands to change the logs folder to `crons/`

Currently only used by `payments.management.CronLoggerMixin`

Usage:

```

>>> config_logger('someapp', 'otherlogger', 'mylogger', log_dir='/full/path/to/
->log/folder')

```

(continues on next page)

**Parameters**

- **logger\_names** (*str*) – List of logger names to replace logging config for (see LOGGER\_NAMES)
- **log\_dir** (*str*) – Fully qualified path. Set each logger’s timed\_file log directory to this

**Returns** `logging.Logger` instance of BASE\_LOGGER

**3.1.1.1.1.5 Module contents****3.1.1.2 Submodules****3.1.1.3 steemengine.helpers module**

Various helper functions for use in CryptoToken Converter.

Copyright:

```
+=====+
|               © 2019 Privex Inc.               |
|               https://www.privex.io             |
+=====+
|
|      CryptoToken Converter                      |
|
|      Core Developer(s):                        |
|
|      (+)  Chris (@someguy123) [Privex]         |
|
+=====+
```

`steemengine.helpers.decrypt_str` (*data: Union[str, bytes], key: Union[str, bytes] = None*) → `str`  
Decrypts data previously encrypted using `encrypt_str()` with the same Fernet compatible key, and returns the decrypted version as a string.

The key cannot just be a random “password”, it must be a 32-byte key encoded with URL Safe base64. Use the management command `./manage.py generate_key` to create a Fernet compatible encryption key.

Under the hood, Fernet uses AES-128 CBC to encrypt the data, with PKCS7 padding and HMAC\_SHA256 authentication.

If the `key` parameter isn’t passed, or is empty (`None` / `""`), then it will attempt to fall back to `settings. ENCRYPT_KEY` - if that’s also empty, `EncryptKeyMissing` will be raised.

**Parameters**

- **data** (*str*) – The base64 encoded data to be decrypted, in the form of either a `str` or `bytes`.
- **key** (*str*) – A Fernet encryption key (base64) for decryption, if blank, will fall back to `settings.ENCRYPT_KEY`

**Raises**

- **EncryptKeyMissing** – Either no key was passed, or something is wrong with the key.
- **EncryptionError** – Something went wrong while attempting to decrypt the data

**Return str decrypted\_data** The decrypted data as a string

`steemengine.helpers.empty(v, zero=False, itr=False) → bool`

Quickly check if a variable is empty or not. By default only '' and None are checked, use *itr* and *zero* to test for empty iterable's and zeroed variables.

Returns True if a variable is None or '', returns False if variable passes the tests

#### Parameters

- **v** – The variable to check if it's empty
- **zero** – if zero=True, then return True if the variable is 0
- **itr** – if itr=True, then return True if the variable is [], {}, or is an iterable and has 0 length

**Return bool is\_blank** True if a variable is blank (None, '', 0, [] etc.)

**Return bool is\_blank** False if a variable has content (or couldn't be checked properly)

`steemengine.helpers.encrypt_str(data: Union[str, bytes], key: Union[str, bytes] = None) → str`

Encrypts a piece of data *data* passed as a string or bytes using Fernet with the passed 32-bit symmetric encryption key *key*. Outputs the encrypted data as a Base64 string for easy storage.

The *key* cannot just be a random "password", it must be a 32-byte key encoded with URL Safe base64. Use the management command `./manage.py generate_key` to create a Fernet compatible encryption key.

Under the hood, Fernet uses AES-128 CBC to encrypt the data, with PKCS7 padding and HMAC\_SHA256 authentication.

If the *key* parameter isn't passed, or is empty (None / ""), then it will attempt to fall back to `settings. ENCRYPT_KEY` - if that's also empty, `EncryptKeyMissing` will be raised.

#### Parameters

- **data** (*str*) – The data to be encrypted, in the form of either a str or bytes.
- **key** (*str*) – A Fernet encryption key (base64) to be used, if left blank will fall back to `settings.ENCRYPT_KEY`

#### Raises

- **EncryptKeyMissing** – Either no key was passed, or something is wrong with the key.
- **EncryptionError** – Something went wrong while attempting to encrypt the data

**Return str encrypted\_data** The encrypted version of the passed data as a base64 encoded string.

`steemengine.helpers.get_fernet(key: Union[str, bytes] = None) → cryptography.fernet.Fernet`

Used internally for getting Fernet instance with auto-fallback to `settings.ENCRYPT_KEY` and exception handling.

**Parameters** **key** (*str*) – Base64 Fernet symmetric key for en/decrypting data. If empty, will fall-back to `settings.ENCRYPT_KEY`

**Raises** **EncryptKeyMissing** – Either no key was passed, or something is wrong with the key.

**Return Fernet f** Instance of Fernet using passed *key* or `settings.ENCRYPT_KEY` for encryption.

`steemengine.helpers.is_encrypted(data: Union[str, bytes], key: Union[str, bytes] = None) → bool`

Returns True if the passed *data* appears to be encrypted. Can only verify encryption if the same *key* that was used to encrypt the data is passed.

#### Parameters

- **data** (*str*) – The data to check for encryption, either as a string or bytes

- **key** (*str*) – Base64 encoded Fernet symmetric key for decrypting data. If empty, fallback to settings.ENCRIPT\_KEY

**Raises** **EncryptKeyMissing** – Either no key was passed, or something is wrong with the key.

**Return** **bool is\_encrypted** True if the data is encrypted, False if it's not encrypted or wrong key used.

`steemengine.helpers.random_str (size=50, chars='abcdefghijklmnopqrstuvwxyz2345679ACDEFGHJKLMNPRSTWXYZ')`

### 3.1.1.4 steemengine.urls module

steemengine URL Configuration

**The *urlpatterns* list routes URLs to views. For more information please see:** <https://docs.djangoproject.com/en/2.1/topics/http/urls/>

Copyright:

```
+=====+
|               © 2019 Privex Inc.               |
|               https://www.privex.io              |
+=====+
|               CryptoToken Converter              |
|               Core Developer(s):                 |
|               (+)  Chris (@someguy123) [Privex]  |
|               +=====+                         |
+=====+
```

`steemengine.urls.path (route, view, kwargs=None, name=None, *, Pattern=<class 'django.urls.resolvers.RoutePattern'>)`

`steemengine.urls.re_path (route, view, kwargs=None, name=None, *, Pattern=<class 'django.urls.resolvers.RegexPattern'>)`

### 3.1.1.5 steemengine.wsgi module

WSGI config for steemengine project.

It exposes the WSGI callable as a module-level variable named `application`.

For more information on this file, see <https://docs.djangoproject.com/en/2.1/howto/deployment/wsgi/>

### 3.1.1.6 Module contents

## 3.2 *payments* module

### 3.2.1 Coin Handlers

#### 3.2.1.1 Subpackages

##### 3.2.1.1.1 Bitcoind Coin Handler

##### 3.2.1.1.1.1 Module contents

#### Bitcoind-based Coin Handler

This python module is a **Coin Handler** for Privex's CryptoToken Converter, designed to handle all required functionality for both receiving and sending any cryptocurrency which has a coin daemon that has a JSONRPC API backwards compatible with *bitcoind*.

It will automatically handle any `payments.models.Coin` which has its type set to `bitcoind`

#### Coin object settings:

For each coin you intend to use with this handler, you should configure it as such:

Coin Key	Description
<code>coin_type</code>	This should be set to <code>Bitcoind RPC compatible crypto</code> (db value: <code>bitcoind</code> )
<code>setting_host</code>	The IP or hostname for the daemon. If not specified, defaults to <code>127.0.0.1 / localhost</code>
<code>setting_port</code>	The RPC port for the daemon. If not specified, defaults to <code>8332</code>
<code>setting_user</code>	The rpcuser for the daemon. Generally <b>MUST</b> be specified.
<code>setting_pass</code>	The rpcpassword for the daemon. Generally <b>MUST</b> be specified
<code>setting_json</code>	A JSON string for optional extra config (see below)

Extra JSON (Handler Custom) config options:

- `confirms_needed` Default 0; Amount of confirmations needed before loading a TX
- `use_trusted` Default: True; If enabled, TXs returned from the daemon with `'trusted':true` will always be accepted at 0 confs regardless of `confirms_needed`
- `string_amt` Default: True; If true, when sending coins, a `Decimal` will be used (as a string). This can cause problems with older coins such as Dogecoin, so for older coins that need floats, set this to False.

#### Django Settings:

If you'd rather not store the RPC details in the database, you may specify them in Django's `settings.py`.

If a coin symbol is specified in `settings.COIND_RPC` they will be used exclusively, and any handler settings on the Coin object will be ignored.

If a settings key isn't specified, the default is the same as shown for coin object settings.

Example `COIND_RPC` Setting:

```
COIND_RPC = {
    "BTC": {
        'user': 'bitcoinrpc',
```

(continues on next page)

(continued from previous page)

```
'password': 'SuperSecurePass',
'host':     '127.0.0.1',
'port':     8332,
'confirms_needed': 0,
'string_amt': True,
'use_trusted': True
}
}
```

### Copyright:

```
+=====+
|                © 2019 Privex Inc.                |
|                https://www.privex.io              |
+=====+
|                CryptoToken Converter              |
|                Core Developer(s):                 |
|                (+)  Chris (@someguy123) [Privex]   |
+=====+
```

`payments.coin_handlers.Bitcoin.reload()`

Reload's the provides property for the loader and manager from the DB.

By default, as there are many coins that use a direct fork of bitcoind, our classes can provide for any models .  
Coin by scanning for coins with the type bitcoind. This saves us from hard coding specific coin symbols.

### 3.2.1.1.2 Submodules

### 3.2.1.1.3 BitcoinLoader module

### Copyright:

```
+=====+
|                © 2019 Privex Inc.                |
|                https://www.privex.io              |
+=====+
|                CryptoToken Converter              |
|                Core Developer(s):                 |
|                (+)  Chris (@someguy123) [Privex]   |
+=====+
```

**class** `payments.coin_handlers.Bitcoin.BitcoinLoader.BitcoinLoader` (*symbols*)  
Bases: `payments.coin_handlers.base.BatchLoader.BatchLoader`, `payments.coin_handlers.Bitcoin.BitcoinMixin.BitcoinMixin`

BitcoinLoader - Despite the name, loads TXs from any coin that has a bitcoind-compatible JsonRPC API

Known to work with: bitcoind, litecoind, dogecoin

**Copyright:**

```

+=====+
|               © 2019 Privex Inc.               |
|               https://www.privex.io             |
+=====+
|
|               CryptoToken Converter             |
|
|               Core Developer(s):                |
|
|               (+)  Chris (@someguy123) [Privex]  |
|
+=====+

```

For the **required Django settings**, please see the module docstring in `coin_handlers.Bitcoin`

**clean\_txs** (*symbol: str, transactions: Iterable[dict], account: str = None*) → Generator[dict, None, None]

Filters a list of transactions *transactions* as required, yields dict's conforming with `models.Deposit`

- Filters out transactions that are not marked as 'receive'
- Filters out mining transactions
- Filters by address if *account* is specified
- Filters out transactions that don't have enough confirms, and are not reported as 'trusted'

**Parameters**

- **symbol** – Symbol of coin being cleaned
- **transactions** – A list<dict> or generator producing dict's
- **account** – If not None, only return TXs sent to this address.

**Return Generator<dict>** A generator outputting dictionaries formatted as below

Output Format:

```

{
    txid:str, coin:str (symbol), vout:int,
    tx_timestamp:datetime, address:str, amount:Decimal
}

```

**load\_batch** (*symbol, limit=100, offset=0, account=None*)

Loads a batch of transactions for *symbol* in their original format into *self.transactions*

**Parameters**

- **symbol** (*str*) – The coin symbol to load TXs for
- **limit** (*int*) – The amount of transactions to load
- **offset** (*int*) – The amount of most recent TXs to skip (for pagination)
- **account** (*str*) – NOT USED BY THIS LOADER

**provides**

Dynamically populated by `Bitcoin.__init__`

**rpcs = {}**

For each coin connection specified in *settings.COIND\_RPC*, we map it's symbol to an instantiated instance of BitcoinRPC - stored as a static property, ensuring we don't have to constantly re-create them.

**settings**

To ensure we always get fresh settings from the DB after a reload, self.settings gets `_prep_settings()`

### 3.2.1.1.1.4 BitcoinManager module

**Copyright:**

```

+=====+
|                © 2019 Privex Inc.                |
|                https://www.privex.io              |
+=====+
|
|                CryptoToken Converter              |
|
|                Core Developer(s):                 |
|
|                (+)  Chris (@someguy123) [Privex]   |
|
+=====+

```

**class** `payments.coin_handlers.Bitcoin.BitcoinManager.BitcoinManager` (*symbol:*

*str*)  
 Bases: `payments.coin_handlers.base.BaseManager.BaseManager`, `payments.coin_handlers.Bitcoin.BitcoinMixin.BitcoinMixin`

BitcoinManager - Despite the name, handles sending, balance, and deposit addresses for any coin that has a bitcoind-compatible JsonRPC API

Known to work with: bitcoind, litecoind, dogecoin

**Copyright:**

```

+=====+
|                © 2019 Privex Inc.                |
|                https://www.privex.io              |
+=====+
|
|                CryptoToken Converter              |
|
|                Core Developer(s):                 |
|
|                (+)  Chris (@someguy123) [Privex]   |
|
+=====+

```

For the **required Django settings**, please see the module docstring in `coin_handlers.Bitcoin`

**address\_valid** (*address*) → bool

If *address* is determined to be valid by the coind RPC, will return True. Otherwise False.

**balance** (*address: str = None, memo: str = None, memo\_case: bool = False*) → decimal.Decimal

Get the total amount received by an address, or the balance of the wallet if address not specified.

**Parameters**

- **address** – Crypto address to get balance for, if None, returns whole wallet balance



- **memo** – NOT USED BY THIS MANAGER
- **memo\_case** – NOT USED BY THIS MANAGER

**Returns** Decimal(balance)

**get\_deposit()** → tuple

Returns a deposit address for this symbol :return tuple: A tuple containing ('address', crypto\_address)

**health()** → Tuple[str, tuple, tuple]

Return health data for the passed symbol.

Health data will include: Symbol, Status, Current Block, Node Version, Wallet Balance, and number of p2p connections (all as strings)

**Return tuple health\_data** (manager\_name:str, headings:list/tuple, health\_data:list/tuple,)

**health\_test()** → bool

Check if the coin daemon is up or not, by requesting basic information such as current block and version.

**Return bool** True if the coin daemon appears to be working, False if it's not

**provides**

Dynamically populated by Bitcoin.\_\_init\_\_

**rpcs = {}**

For each coin connection specified in *settings.COIND\_RPC*, we map it's symbol to an instantiated instance of BitcoinRPC - stored as a static property, ensuring we don't have to constantly re-create them.

**send(amount, address, memo=None, from\_address=None, trigger\_data=None)** → dict

Send the amount *amount* of *self.symbol* to a given address.

Example - send 0.1 LTC to LVXXmgcVYBZAuiJM3V99uG48o3yG89h2Ph

```
>>> s = BitcoinManager('LTC')
>>> s.send(address='LVXXmgcVYBZAuiJM3V99uG48o3yG89h2Ph', amount=Decimal('0.1
↳ '))
```

### Parameters

- **amount** (*Decimal*) – Amount of coins to send, as a Decimal()
- **address** – Address to send the coins to
- **from\_address** – NOT USED BY THIS MANAGER
- **memo** – NOT USED BY THIS MANAGER

### Raises

- **AccountNotFound** – The destination *address* isn't valid
- **NotEnoughBalance** – The wallet does not have enough balance to send this amount.

**Return dict** Result Information

Format:

```
{
    txid:str - Transaction ID - None if not known,
    coin:str - Symbol that was sent,
    amount:Decimal - The amount that was sent (after fees),
    fee:Decimal - TX Fee that was taken from the amount,
    from:str - The account/address the coins were sent from,
```

(continues on next page)

(continued from previous page)

```
    send_type:str          - Should be statically set to "send"
}
```

### **setting**

Retrieve only our symbol from self.settings for convenience

### **settings**

To ensure we always get fresh settings from the DB after a reload, self.settings gets `_prep_settings()`

#### 3.2.1.1.1.5 BitcoinMixin module

##### Copyright:

```
+=====+
|                © 2019 Privex Inc.                |
|                https://www.privex.io              |
+=====+
|
|    CryptoToken Converter
|
|    Core Developer(s):
|
|    (+)  Chris (@someguy123) [Privex]
|
+=====+
```

**class** `payments.coin_handlers.Bitcoin.BitcoinMixin.BitcoinMixin`

Bases: `object`

BitcoinMixin - shared code used by both `Bitcoin.BitcoinLoader` and `Bitcoin.BitcoinManager`

##### Copyright:

```
+=====+
|                © 2019 Privex Inc.                |
|                https://www.privex.io              |
+=====+
|
|    CryptoToken Converter
|
|    Core Developer(s):
|
|    (+)  Chris (@someguy123) [Privex]
|
+=====+
```

### **all\_coins**

Since this is a Mixin, it may be `self.coin: Coin`, or `self.coins: List[Coin]`. This property detects whether we have a single coin, or multiple, and returns them as a dict.

**Return dict coins** A dict<str,Coin> of supported coins, mapped by symbol

### 3.2.1.1.2 SteemEngine Coin Handler

#### 3.2.1.1.2.1 Module contents

##### SteemEngine Coin Handler

This python module is a **Coin Handler** for Privex's CryptoToken Converter, designed to handle all required functionality for both receiving and sending tokens on the **SteemEngine** network.

It will automatically handle any `payments.models.Coin` which has its type set to `steemengine`

##### Copyright:

```

+=====+
|                © 2019 Privex Inc.                |
|                https://www.privex.io              |
+=====+
|
|                CryptoToken Converter              |
|
|                Core Developer(s):                 |
|
|                (+)  Chris (@someguy123) [Privex]   |
|
+=====+

```

`payments.coin_handlers.SteemEngine.reload()`

Reload's the provides property for the loader and manager from the DB.

By default, since new tokens are constantly being created for SteemEngine, our classes can provide for any `models.Coin` by scanning for coins with the type `steemengine`. This saves us from hard coding specific coin symbols.

#### 3.2.1.1.2.2 Submodules

#### 3.2.1.1.2.3 SteemEngineLoader module

##### Copyright:

```

+=====+
|                © 2019 Privex Inc.                |
|                https://www.privex.io              |
+=====+
|
|                CryptoToken Converter              |
|
|                Core Developer(s):                 |
|
|                (+)  Chris (@someguy123) [Privex]   |
|
+=====+

```

**class** `payments.coin_handlers.SteemEngine.SteemEngineLoader.SteemEngineLoader` (*symbols*)

Bases: `payments.coin_handlers.base.BaseLoader.BaseLoader`

This class handles loading transactions for the **SteemEngine** network, and can support almost any token on SteemEngine.

## Copyright:

```

+=====+
|               © 2019 Privex Inc.               |
|               https://www.privex.io             |
+=====+
|
|               CryptoToken Converter             |
|
|               Core Developer(s):                |
|
|               (+)  Chris (@someguy123) [Privex]  |
|
+=====+

```

**clean\_txs** (*account: str, symbol: str, transactions: Iterable[dict]*) → Generator[dict, None, None]

Filters a list of transactions by the receiving account, yields dict's conforming with [payments.models.Deposit](#)

### Parameters

- **account** (*str*) – The 'to' account to filter by
- **symbol** (*str*) – The symbol of the token being filtered
- **transactions** (*list<dict>*) – A list<dict> of transactions to filter

**Returns** A generator yielding dict's conforming to [payments.models.Deposit](#)

**list\_txs** (*batch=100*) → Generator[dict, None, None]

Get transactions for all coins in *self.coins* where the 'to' field matches coin.our\_account If *load()* hasn't been ran already, it will automatically call *self.load()* :param batch: Amount of transactions to load per batch :return: Generator yielding dict's that conform to *models.Deposit*

**load** (*tx\_count=1000*)

The load function should prepare your loader, by either importing all of the data required for filtering, or setting up a generator for the *list\_txs()* method to load them paginated.

It does NOT return anything, it simply creates any connections required, sets up generator functions if required for paginating the data, and/or pre-loads the first batch of transaction data.

**Parameters tx\_count** – The total amount of transactions that should be loaded PER SYM-BOL, most recent first.

**Returns** None

**load\_batch** (*account, symbol, limit=100, offset=0, retry=0*)

Load SteemEngine transactions for account/symbol into *self.transactions* with automatic retry on error

### provides

This attribute is automatically generated by scanning for *models.Coins* with the type *steemengine*. This saves us from hard coding specific coin symbols. See *\_\_init\_\_.py* for populating code.

## 3.2.1.1.2.4 SteemEngineManager module

## Copyright:

```

+=====+
|               © 2019 Privex Inc.               |
|               https://www.privex.io             |

```

(continues on next page)

(continued from previous page)

```

+=====+
|                                     |
|      CryptoToken Converter         |
|                                     |
|      Core Developer(s):           |
|                                     |
|      (+)  Chris (@someguy123) [Privex] |
|                                     |
+=====+

```

**class** `payments.coin_handlers.SteemEngine.SteemEngineManager.SteemEngineManager` (*symbol: str*)

Bases: `payments.coin_handlers.base.BaseManager.BaseManager`

This class handles various operations for the **SteemEngine** network, and supports almost any token on SteemEngine.

It handles:

- Validating source/destination accounts
- Checking the balance for a given account, as well as the total amount received with a certain `memo`
- Issuing tokens to users
- Sending tokens to users

**Copyright:**

```

+=====+
|                                     |
|      © 2019 Privex Inc.             |
|      https://www.privex.io          |
+=====+
|                                     |
|      CryptoToken Converter         |
|                                     |
|      Core Developer(s):           |
|                                     |
|      (+)  Chris (@someguy123) [Privex] |
|                                     |
+=====+

```

**address\_valid** (*address*) → bool

If an account ( *address* param) exists on Steem, will return True. Otherwise False.

**balance** (*address: str = None, memo: str = None, memo\_case: bool = False*) → decimal.Decimal

Get token balance for a given Steem account, if `memo` is given - get total symbol amt received with this `memo`.

**Parameters**

- **address** – Steem account to get balance for, if not set, uses `self.coin.our_account`
- **memo** – If not None, get total *self.symbol* received with this `memo`.
- **memo\_case** – Case sensitive memo search

**Returns** Decimal(balance)

**get\_deposit** () → tuple

Returns the deposit account for this symbol

**Return tuple** A tuple containing ('account', receiving\_account). The memo must be generated by the calling function.

**health()** → Tuple[str, tuple, tuple]

Return health data for the passed symbol.

Health data will include: Symbol, Status, Current Block, Node Version, Wallet Balance, and number of p2p connections (all as strings)

**Return tuple health\_data** (manager\_name:str, headings:list/tuple, health\_data:list/tuple,)

**health\_test()** → bool

Check if the SteemEngine API and Steem node works or not, by requesting basic information such as the token metadata, and checking if our sending/receiving account exists on Steem.

**Return bool** True if SteemEngine and Steem appear to be working, False if either is broken.

**issue** (amount: decimal.Decimal, address: str, memo: str = None, trigger\_data=None) → dict

Issue (create/print) tokens to a given address/account, optionally specifying a memo if supported

Example - Issue 5.10 SGTk to @privex

```
>>> s = SteemEngineManager('SGTK')
>>> s.issue(address='privex', amount=Decimal('5.10'))
```

### Parameters

- **amount** (*Decimal*) – Amount of tokens to issue, as a Decimal()
- **address** – Address or account to issue the tokens to
- **memo** – (ignored) Cannot issue tokens with a memo on SteemEngine

### Raises

- **IssuerKeyError** – Cannot issue because we don't have authority to (missing key etc.)
- **IssueNotSupported** – Class does not support issuing, or requested symbol cannot be issued.
- **AccountNotFound** – The requested account/address doesn't exist

**Return dict** Result Information

Format:

```
{
    txid:str - Transaction ID - None if not known,
    coin:str - Symbol that was sent,
    amount:Decimal - The amount that was sent (after fees),
    fee:Decimal - TX Fee that was taken from the amount,
    from:str - The account/address the coins were issued from,
    send_type:str - Should be statically set to "issue"
}
```

### provides

This attribute is automatically generated by scanning for models.Coins with the type steemengine. This saves us from hard coding specific coin symbols. See \_\_init\_\_.py for populating code.

**send** (amount, address, memo=None, from\_address=None, trigger\_data=None) → dict

Send tokens to a given address/account, optionally specifying a memo if supported

Example - send 1.23 SGTk from @someguy123 to @privex with memo 'hello'

```
>>> s = SteemEngineManager('SGTK')
>>> s.send(from_address='someguy123', address='privex', amount=Decimal('1.23
↳ '), memo='hello')
```

### Parameters

- **amount** (*Decimal*) – Amount of tokens to send, as a *Decimal()*
- **address** – Account to send the tokens to
- **from\_address** – Account to send the tokens from
- **memo** – Memo to send tokens with (if supported)

### Raises

- **AttributeError** – When both *from\_address* and *self.coin.our\_account* are blank.
- **ArithmeticError** – When the amount is lower than the lowest amount allowed by the token's precision
- **AuthorityMissing** – Cannot send because we don't have authority to (missing key etc.)
- **AccountNotFound** – The requested account/address doesn't exist
- **TokenNotFound** – When the requested token *symbol* does not exist
- **NotEnoughBalance** – The account *from\_address* does not have enough balance to send this amount.

### Return dict Result Information

Format:

```
{
    txid:str - Transaction ID - None if not known,
    coin:str - Symbol that was sent,
    amount:Decimal - The amount that was sent (after fees),
    fee:Decimal - TX Fee that was taken from the amount,
    from:str - The account/address the coins were sent from,
    send_type:str - Should be statically set to "send"
}
```

**send\_or\_issue** (*amount, address, memo=None, trigger\_data=None*) → dict

Attempt to send an amount to an address/account, if not enough balance, attempt to issue it instead. You may override this method if needed.

### Parameters

- **amount** (*Decimal*) – Amount of coins/tokens to send/issue, as a *Decimal()*
- **address** – Address or account to send/issue the coins/tokens to
- **memo** – Memo to send/issue coins/tokens with (if supported)
- **trigger\_data** (*dict*) – Metadata related to this issue transaction (e.g. the deposit that triggered this)

### Raises

- **IssuerKeyError** – Cannot issue because we don't have authority to (missing key etc.)

- **IssueNotSupported** – Class does not support issuing, or requested symbol cannot be issued.
- **AccountNotFound** – The requested account/address doesn't exist

### Return dict Result Information

Format:

```
dict {
  txid:str          - Transaction ID - None if not known,
  coin:str          - Symbol that was sent,
  amount:Decimal    - The amount that was sent (after fees),
  fee:Decimal       - TX Fee that was taken from the amount,
  from:str          - The account(s)/address(es) the coins were sent from. if
  ↳ more than one, comma separated.
                    If it's not possible to determine easily, set this to None.
  send_type:str     - Should be set to "send" if the coins were sent, or "issue"
  ↳ if the coins were issued.
}
```

### 3.2.1.1.3 Steem Coin Handler

#### 3.2.1.1.3.1 Module contents

#### Steem Coin Handler

This python module is a **Coin Handler** for Privex's CryptoToken Converter, designed to handle all required functionality for both receiving and sending tokens on the **Steem** network.

It will automatically handle any `payments.models.Coin` which has its type set to `steembase`

#### Coin object settings:

For each `payments.models.Coin` you intend to use with this handler, you should configure it as such:

Coin Key	Description
coin_type	This should be set to Steem Network (or compatible fork) (db value: steembase)
our_account	This should be set to the username of the account you want to use for receiving/sending
setting_json	A JSON string for optional extra config (see below)

Extra JSON (Handler Custom) config options:

- `rpcs` - A JSON list<str> of RPC nodes to use, with a full HTTP/HTTPS URL. If this is not specified, Beem will automatically try to use the best available RPC node for the Steem network.
- `pass_store` - Generally you do not need to touch this. It controls where Beem will look for the wallet password. It defaults to `environment`

Example JSON custom config:

```
{
  "rpcs": [
    "https://steemd.privex.io",
```

(continues on next page)



(continued from previous page)

```

    "https://api.steemit.com",
    "https://api.steem.house"
  ],
  "pass_store": "environment"
}

```

**Copyright:**

```

+=====+
|                © 2019 Privex Inc.                |
|                https://www.privex.io                |
+=====+
|
|      CryptoToken Converter      |
|
|      Core Developer(s):        |
|
|      (+)  Chris (@someguy123) [Privex]              |
|
+=====+

```

`payments.coin_handlers.Steem.reload()`

Reload's the provides property for the loader and manager from the DB.

By default, since new Steem forks are constantly being created, our classes can provide for any `models.Coin` by scanning for coins with the type `steembase`. This saves us from hard coding specific coin symbols.

**3.2.1.1.3.2 Submodules****3.2.1.1.3.3 SteemLoader module****Copyright:**

```

+=====+
|                © 2019 Privex Inc.                |
|                https://www.privex.io                |
+=====+
|
|      CryptoToken Converter      |
|
|      Core Developer(s):        |
|
|      (+)  Chris (@someguy123) [Privex]              |
|
+=====+

```

**class** `payments.coin_handlers.Steem.SteemLoader.SteemLoader` (*symbols*)

Bases: `payments.coin_handlers.base.BaseLoader.BaseLoader`, `payments.coin_handlers.Steem.SteemMixin.SteemMixin`

SteemLoader - Loads transactions from the Steem network

Designed for the Steem Network with SBD and STEEM support. May or may not work with other Graphene coins.

**Copyright:**

```

+=====+
|               © 2019 Privex Inc.               |
|               https://www.privex.io             |
+=====+
|
|           CryptoToken Converter                 |
|
|           Core Developer(s):                   |
|
|           (+)  Chris (@someguy123) [Privex]     |
|
+=====+

```

For **additional settings**, please see the module docstring in `coin_handlers.Steem`

**static clean\_tx** (*tx: dict, symbol: str, account: str, memo: str = None, memo\_case: bool = False*)  
 → Optional[dict]

Filters an individual transaction. See `clean_txs()` for info

**clean\_txs** (*symbol: str, transactions: Iterable[dict], account: str = None*) → Generator[dict, None, None]

Filters a list of transactions *transactions* as required, yields dict's conforming with `models.Deposit`

- Filters out transactions that are not marked as 'receive'
- Filters out mining transactions
- Filters by address if *account* is specified
- Filters out transactions that don't have enough confirms, and are not reported as 'trusted'

### Parameters

- **symbol** – Symbol of coin being cleaned
- **transactions** – A list<dict> or generator producing dict's
- **account** – If not None, only return TXs sent to this address.

**Return Generator<dict>** A generator outputting dictionaries formatted as below

Output Format:

```

{
  txid:str, coin:str (symbol), vout:int,
  tx_timestamp:datetime, address:str, amount:Decimal
}

```

**list\_txs** (*batch=0*) → Generator[dict, None, None]

The `list_txs` function processes the transaction data from `load()`, as well as handling any pagination, if it's required (e.g. only retrieve *batch* transactions at a time from the data source)

It should first check that `load()` has been ran if it's required, if the data required has not been loaded, it should call `self.load()`

To prevent memory leaks, this must be a generator function.

Below is an example of a generator function body, it loads *batch* transactions from the full transaction list, pretends to processes them into *txs*, yields them, then loads another batch after the calling function has iterated over the current *txs*

```

>>> t = self.transactions # All transactions
>>> b = batch
>>> finished = False
>>> offset = 0
>>> # To save memory, process 100 transactions per iteration, and yield them.
↳(generator)
>>> while not finished:
>>>     txs = [] # Processed transactions
>>>     # If there are less remaining TXs than batch size, get remaining txs.
↳and finish.
>>>     if (len(t) - offset) < batch:
>>>         finished = True
>>>     # Do some sort-of processing on the tx to make it conform to.
↳`Deposit`, then append to txs
>>>     for tx in t[offset:offset + batch]:
>>>         txs.append(tx)
>>>         offset += b
>>>     for tx in txs:
>>>         yield tx
>>>     # At this point, the current batch is exhausted. Destroy the tx array.
↳to save memory.
>>>     del txs

```

**Parameters** `batch` (*int*) – Amount of transactions to process/load per each batch

**Returns Generator** A generator returning dictionaries that can be imported into models.

Deposit

Dict format:

```
{txid:str, coin:str (symbol), vout:int, tx_timestamp:datetime,
 address:str, from_account:str, to_account:str, memo:str, amount:Decimal}
```

vout is optional. One of either {from\_account, to\_account, memo} OR {address} must be included.

**load** (*tx\_count=10000*)

The load function should prepare your loader, by either importing all of the data required for filtering, or setting up a generator for the `list_txs()` method to load them paginated.

It does NOT return anything, it simply creates any connections required, sets up generator functions if required for paginating the data, and/or pre-loads the first batch of transaction data.

**Parameters** `tx_count` – The total amount of transactions that should be loaded PER SYMBOL, most recent first.

**Returns** None

**provides**

This attribute is automatically generated by scanning for models.Coin s with the type steembase. This saves us from hard coding specific coin symbols. See `__init__.py` for populating code.

**settings**

To ensure we always get fresh settings from the DB after a reload

### 3.2.1.1.3.4 SteemManager module

Copyright:

```
+=====+
|               © 2019 Privex Inc.               |
|               https://www.privex.io             |
+=====+
|
|           CryptoToken Converter                 |
|
|           Core Developer(s) :                  |
|
|           (+)  Chris (@someguy123) [Privex]     |
|
+=====+
```

**class** `payments.coin_handlers.Steem.SteemManager.SteemManager` (*symbol: str*)  
Bases: `payments.coin_handlers.base.BaseManager.BaseManager`, `payments.coin_handlers.Steem.SteemMixin.SteemMixin`

This class handles various operations for the **Steem** network, and supports both STEEM and SBD.

It may or may not work with other Graphene coins, such as GOLOS / Whaleshares.

It handles:

- Validating source/destination accounts
- Checking the balance for a given account, as well as the total amount received with a certain memo
- Health checking
- Sending assets to users

**Copyright:**

```
+=====+
|               © 2019 Privex Inc.               |
|               https://www.privex.io             |
+=====+
|
|           CryptoToken Converter                 |
|
|           Core Developer(s) :                  |
|
|           (+)  Chris (@someguy123) [Privex]     |
|
+=====+
```

**address\_valid** (*address*) → bool

If an account exists on Steem, will return True. Otherwise False.

**Parameters** **address** – Steem account to check existence of

**Return bool** True if account exists, False if it doesn't

**balance** (*address: str = None, memo: str = None, memo\_case: bool = False*) → decimal.Decimal

Get token balance for a given Steem account, if memo is given - get total symbol amt received with this memo.

**Parameters**

- **address** – Steem account to get balance for, if not set, uses `self.coin.our_account`
- **memo** – If not None, get total *self.symbol* received with this memo.

- **memo\_case** – Case sensitive memo search

**Returns** Decimal(balance)

**get\_deposit()** → tuple

Returns the deposit account for this symbol

**Return tuple** A tuple containing ('account', receiving\_account). The memo must be generated by the calling function.

**health()** → Tuple[str, tuple, tuple]

Return health data for the passed symbol.

Health data will include: 'Symbol', 'Status', 'Coin Name', 'API Node', 'Head Block', 'Block Time', 'RPC Version', 'Our Account', 'Our Balance' (all strings)

**Return tuple health\_data** (manager\_name:str, headings:list/tuple, health\_data:list/tuple,)

**health\_test()** → bool

Check if our Steem node works or not, by requesting basic information such as the current block + time, and checking if our sending/receiving account exists on Steem.

**Return bool** True if Steem appears to be working, False if it seems to be broken.

**provides**

This attribute is automatically generated by scanning for `models.Coins` with the type `steembase`. This saves us from hard coding specific coin symbols. See `__init__.py` for populating code.

**send**(amount: decimal.Decimal, address: str, from\_address: str = None, memo=None, trigger\_data=None) → dict

Send a supported currency to a given address/account, optionally specifying a memo if supported

Example - send 1.23 STEEM from @someguy123 to @privex with memo 'hello'

```
>>> s = SteemManager('STEEM')
>>> s.send(from_address='someguy123', address='privex', amount=Decimal('1.23'), memo='hello')
```

**Parameters**

- **amount** (*Decimal*) – Amount of currency to send, as a `Decimal()`
- **address** – Account to send the currency to
- **from\_address** – Account to send the currency from
- **memo** – Memo to send currency with

**Raises**

- **AttributeError** – When both *from\_address* and *self.coin.our\_account* are blank.
- **ArithmeticError** – When the amount is lower than the lowest amount allowed by the asset's precision
- **AuthorityMissing** – Cannot send because we don't have authority to (missing key etc.)
- **AccountNotFound** – The requested account doesn't exist
- **NotEnoughBalance** – The account *from\_address* does not have enough balance to send this amount.

**Return dict** Result Information

Format:

```
{
    txid:str - Transaction ID - None if not known,
    coin:str - Symbol that was sent,
    amount:Decimal - The amount that was sent (after fees),
    fee:Decimal - TX Fee that was taken from the amount,
    from:str - The account/address the coins were sent from,
    send_type:str - Should be statically set to "send"
}
```

### 3.2.1.1.4 Coin Handler Base Classes

#### 3.2.1.1.4.1 Submodules

#### 3.2.1.1.4.2 BaseLoader

**class** `payments.coin_handlers.base.BaseLoader.BaseLoader` (*symbols: list = None*)  
Bases: `abc.ABC`

BaseLoader - Base class for Transaction loaders

A transaction loader loads incoming transactions from one or more cryptocurrencies or tokens, whether through a block explorer, or through a direct connection to a local RPC node such as steemd or bitcoind using connection settings set by the user in their Django settings.

Transaction loaders must be able to initialise themselves using the following data:

- The coin symbols `self.symbols` passed to the constructor
- The setting\_xxx fields on `self.coin` `payments.models.Coin`
- The Django settings *from `django.conf` import `settings`*
- They should also use the logging instance `settings.LOGGER_NAME`

If your class requires anything to be added to the Coin object settings, or the Django settings file, you should write a comment listing which settings are required, which are optional, and their format/type.

e.g. (Optional) settings.STEEM\_NODE - list of steem RPC nodes, or string of individual node, URL format

They must implement all of the methods in this class, as well as configure the *provides* list to display the tokens/coins that this loader handles.

**list\_txs** (*batch=100*) → Generator[dict, None, None]

The `list_txs` function processes the transaction data from `load()`, as well as handling any pagination, if it's required (e.g. only retrieve *batch* transactions at a time from the data source)

It should first check that `load()` has been ran if it's required, if the data required has not been loaded, it should call `self.load()`

To prevent memory leaks, this must be a generator function.

Below is an example of a generator function body, it loads *batch* transactions from the full transaction list, pretends to processes them into *txs*, yields them, then loads another batch after the calling function has iterated over the current *txs*

```

>>> t = self.transactions # All transactions
>>> b = batch
>>> finished = False
>>> offset = 0
>>> # To save memory, process 100 transactions per iteration, and yield them.
↳(generator)
>>> while not finished:
>>>     txs = [] # Processed transactions
>>>     # If there are less remaining TXs than batch size, get remaining txs.
↳and finish.
>>>     if (len(t) - offset) < batch:
>>>         finished = True
>>>     # Do some sort-of processing on the tx to make it conform to.
↳`Deposit`, then append to txs
>>>     for tx in t[offset:offset + batch]:
>>>         txs.append(tx)
>>>     offset += b
>>>     for tx in txs:
>>>         yield tx
>>>     # At this point, the current batch is exhausted. Destroy the tx array.
↳to save memory.
>>>     del txs

```

**Parameters** `batch` (*int*) – Amount of transactions to process/load per each batch

**Returns Generator** A generator returning dictionaries that can be imported into models.

Deposit

Dict format:

```

{txid:str, coin:str (symbol), vout:int, tx_timestamp:datetime,
 address:str, from_account:str, to_account:str, memo:str, amount:Decimal}

```

vout is optional. One of either {from\_account, to\_account, memo} OR {address} must be included.

**load** (*tx\_count=1000*)

The load function should prepare your loader, by either importing all of the data required for filtering, or setting up a generator for the `list_txs()` method to load them paginated.

It does NOT return anything, it simply creates any connections required, sets up generator functions if required for paginating the data, and/or pre-loads the first batch of transaction data.

**Parameters** `tx_count` – The total amount of transactions that should be loaded PER SYMBOL, most recent first.

**Returns** None

`provides = []`

#### 3.2.1.1.4.3 BaseManager

**class** `payments.coin_handlers.base.BaseManager`.**BaseManager** (*symbol: str*)  
 Bases: `abc.ABC`

BaseManager - Base class for coin/token management

A coin manager handles balance checking, sending, and issuing of one or more cryptocurrencies or tokens, generally through a direct connection to a local/remote RPC node such as steemd or bitcoind using connection settings set by the user in their Django settings.

Coin managers must be able to initialise themselves using the following data:

- The coin symbol `self.symbol` passed to the constructor
- The setting\_xxx fields on `self.coin payments.models.Coin`
- The Django settings *from `django.conf` import settings*

If your class requires anything to be added to the Coin object settings, or the Django settings file, you should write a comment listing which settings are required, which are optional, and their format/type.

e.g. (Optional) settings.STEEM\_NODE - list of steem RPC nodes, or string of individual node, URL format

They must implement all of the methods in this class, set the `can_issue` boolean for detecting if this manager can be used to issue (create/print) tokens/coins, as well as configure the `provides` list to display the tokens/coins that this manager handles.

**address\_valid** (*address*) → bool

A simple boolean method, allowing API requests to validate the destination address/account prior to giving the user deposit details.

**Parameters** **address** – An address or account to send to

**Return bool** Is the *address* valid? True if it is, False if it isn't

**balance** (*address: str = None, memo: str = None, memo\_case: bool = False*) → decimal.Decimal

Return the balance of `self.symbol` for our “wallet”, or a given address/account, optionally filtered by memo

**Parameters**

- **address** – The address or account to get the balance for. If None, return our total wallet (or default account) balance.
- **memo** – If not None (and coin supports memos), return the total balance of a given memo
- **memo\_case** – Whether or not to total memo's case sensitive, or not. False = case-insensitive memo

**Raises** *AccountNotFound* – The requested account/address doesn't exist

**Return Decimal** Decimal() balance of address/account, optionally balance (total received) of a given memo

**can\_issue = False**

If this manager supports issuing (creating/printing) tokens/coins, set this to True

**get\_deposit** () → tuple

**Return tuple** If the coin uses addresses, this method should return a tuple of ('address', coin\_address)

**Return tuple** If the coin uses accounts/memos, this method should return a tuple ('account', receiving\_account) The memo will automatically be generated by the calling function.

**health** () → Tuple[str, tuple, tuple]

Return health data for the passed symbol, e.g. current block height, block time, wallet balance whether the daemon / API is accessible, etc.

It should return a tuple containing the manager name, the headings for a health table, and the health data for the passed symbol (Should include a `symbol` or coin name column)



You may use basic HTML tags in the health data result list, such as `<b>` `<em>` `<u>` and `<span style=""></span>`

**Return tuple health\_data** (manager\_name:str, headings:list/tuple, health\_data:list/tuple,)

**health\_test** () → bool

To reduce the risk of unhandled exceptions by sending code, this method should do some basic checks against the API to test whether the coin daemon / API is responding correctly.

This allows code which calls your send() or issue() method to detect the daemon / API is not working, and then delay sending/issuing until later, instead of marking a convert / withdrawal status to an error.

The method body should be wrapped in a try/except, ensuring there's a non-targeted except which returns False

**Return bool** True if the coin daemon / API appears to be working, False if it's not

**issue** (amount: decimal.Decimal, address: str, memo: str = None, trigger\_data: Union[dict, list] = None) → dict

Issue (create/print) tokens to a given address/account, optionally specifying a memo if supported

#### Parameters

- **amount** (*Decimal*) – Amount of tokens to issue, as a Decimal()
- **address** – Address or account to issue the tokens to
- **memo** – Memo to issue tokens with (if supported)
- **trigger\_data** (*dict*) – Metadata related to this issue transaction (e.g. the deposit that triggered this)

#### Raises

- **IssuerKeyError** – Cannot issue because we don't have authority to (missing key etc.)
- **IssueNotSupported** – Class does not support issuing, or requested symbol cannot be issued.
- **AccountNotFound** – The requested account/address doesn't exist

**Return dict** Result Information

Format:

```
dict {
    txid:str - Transaction ID - None if not known,
    coin:str - Symbol that was sent,
    amount:Decimal - The amount that was sent (after fees),
    fee:Decimal - TX Fee that was taken from the amount,
    from:str - The account/address the coins were issued from.
                If it's not possible to determine easily, set this to None.
    send_type:str - Should be statically set to "issue"
}
```

**orig\_symbol = None**

The original unique database symbol ID

**provides = []**

A list of token/coin symbols in uppercase that this loader supports e.g:

```
provides = ["LTC", "BTC", "BCH"]
```

**send** (*amount*: decimal.Decimal, *address*: str, *from\_address*: str = None, *memo*: str = None, *trigger\_data*: Union[dict, list] = None) → dict

Send tokens to a given address/account, optionally specifying a memo and sender address/account if supported

Your send method should automatically subtract any blockchain transaction fees from the amount sent.

### Parameters

- **amount** (*Decimal*) – Amount of coins/tokens to send, as a Decimal()
- **address** – Address or account to send the coins/tokens to
- **memo** – Memo to send coins/tokens with (if supported)
- **from\_address** – Address or account to send from (if required)
- **trigger\_data** (*dict*) – Metadata related to this send transaction (e.g. the deposit that triggered this)

### Raises

- **AuthorityMissing** – Cannot send because we don't have authority to (missing key etc.)
- **AccountNotFound** – The requested account/address doesn't exist
- **NotEnoughBalance** – Sending account/address does not have enough balance to send

### Return dict Result Information

Format:

```
dict {
  txid:str - Transaction ID - None if not known,
  coin:str - Symbol that was sent,
  amount:Decimal - The amount that was sent (after fees),
  fee:Decimal - TX Fee that was taken from the amount,
  from:str - The account(s)/address(es) the coins were sent from. if
  ↪more than one, comma separated.
      If it's not possible to determine easily, set this to None.
  send_type:str - Should be statically set to "send"
}
```

**send\_or\_issue** (*amount*, *address*, *memo*=None, *trigger\_data*: Union[dict, list] = None) → dict

Attempt to send an amount to an address/account, if not enough balance, attempt to issue it instead. You may override this method if needed.

### Parameters

- **amount** (*Decimal*) – Amount of coins/tokens to send/issue, as a Decimal()
- **address** – Address or account to send/issue the coins/tokens to
- **memo** – Memo to send/issue coins/tokens with (if supported)
- **trigger\_data** (*dict*) – Metadata related to this issue transaction (e.g. the deposit that triggered this)

### Raises

- **IssuerKeyError** – Cannot issue because we don't have authority to (missing key etc.)
- **IssueNotSupported** – Class does not support issuing, or requested symbol cannot be issued.

- **AccountNotFound** – The requested account/address doesn't exist

#### Return dict Result Information

Format:

```
dict {
    txid:str          - Transaction ID - None if not known,
    coin:str          - Symbol that was sent,
    amount:Decimal    - The amount that was sent (after fees),
    fee:Decimal       - TX Fee that was taken from the amount,
    from:str          - The account(s)/address(es) the coins were sent from. if
    ↪more than one, comma separated.
                        If it's not possible to determine easily, set this to None.
    send_type:str     - Should be set to "send" if the coins were sent, or "issue"
    ↪if the coins were issued.
}
```

**symbol = None**

The native coin symbol, e.g. BTC, LTC, etc. (non-unique)

#### 3.2.1.1.4.4 BatchLoader

**class** payments.coin\_handlers.base.BatchLoader.**BatchLoader** (symbols: list = None)

Bases: `payments.coin_handlers.base.BaseLoader.BaseLoader`, `abc.ABC`

BatchLoader - An abstract sub-class of BaseLoader which comes with some pre-written batching/chunking functions

Copyright:

```
+=====+
|                © 2019 Privex Inc.                |
|                https://www.privex.io              |
+=====+
|
|      CryptoToken Converter                        |
|
|      Core Developer(s):                          |
|
|      (+)  Chris (@someguy123) [Privex]            |
|
+=====+
```

This class is designed to save you time from re-writing your own “batching” / “chunking” functions.

Batching / chunking is a memory efficiency technique to prevent RAM leaks causing poor performance or crashes. Instead of loading all 1K - 10K transactions into memory, you load only a small amount of transactions, such as 100 transactions, then you use a Python generator (the yield keyword) to return individual transactions, quietly loading the next “batch” / “chunk” of 100 TXs after the first set has been processed, without interrupting the caller's *for* loop or other iteration.

This allows other functions to iterate over the transactions and process them on the fly, instead of having to load the entire 1-10K transaction list into memory first.

The use of generators throughout this class helps to prevent the problem of RAM leaks due to constant duplication of the transaction list (e.g. `self.transactions`, `self.filtered_txs`, `self.cleaned_txs`), especially when the transaction lists contains thousands of transactions.

To use this class, simply extend it (instead of BaseLoader), and make sure to implement the two abstract methods:

- `load_batch` - Loads and stores a small batch of raw (original format) transactions for a given coin
- `clean_txs` - Filters the loaded TXs, yielding TXs (conformed to be compatible with `models.Deposit`) that were received by us (not sent), and various sanity checks depending on the type of coin.

If your Loader is for a coin which uses an account/memo system, set `self.need_account = True` before calling BatchLoader's constructor, and it will remove coins in `self.symbols/coins` that do not have a non-empty/null `our_account` column.

You're free to override any methods if you need to, just make sure to call this class's constructor `__init__` before/after your own constructor, otherwise some methods may break.

Flow of this class:

```
Transaction loading cron
|
V--> __init__(symbols:list)
|--> load(tx_count:int)
|--> list_txs(batch:int) -> _list_txs(coin:Coin, batch:int)
V                                |--> load_batch(account, symbol, offset)
                                V--> clean_txs(account, symbol, txs)
```

`clean_txs` (`symbol: str, transactions: Iterable[dict], account: str = None`) → Generator[dict, None, None]

Filters a list of transactions as required, yields dict's conforming with `models.Deposit`

Important things when implementing this function:

- Make sure to filter out transactions that were sent from our own wallet/account - otherwise internal transfers will cause problems.
- Make sure each transaction is destined to us
  - If your loader is account-based, make sure to only yield transactions where `tx["to_account"] == account`.
  - If your loader is address-based, make sure that you only return transactions that are being received by our wallet, not being sent from it.
  - \* If `account` isn't `None`, assume that you must yield TXs sent to the given crypto address `account`
- If your loader deals with smart contract networks e.g. ETH, EOS, make sure that you only return transactions valid on the matching smart contract, don't blindly trust the symbol!
- Make sure that every dict that you yield conforms with the return standard shown for `BaseLoader.list_txs()`
- While `transactions` is normally a list<dict> you should assume that it could potentially be a Generator, writing the code Generator-friendly will ensure it can handle both lists and Generator's.

Example:

```
>>> def clean_txs(self, symbol: str, transactions: Iterable[dict],
>>>                 account: str = None) -> Generator[dict, None, None]:
>>>     for tx in transactions:
>>>         try:
>>>             if tx['from'].lower() == 'tokens': continue # Ignore_
↳ token issues
```

(continues on next page)

(continued from previous page)

```

>>>         if tx['from'].lower() == account: continue           # Ignore_
↳ transfers from ourselves.
>>>         if tx['to'].lower() != account.lower(): continue      # If we aren
↳ t the receiver, we don't need it.
>>>         clean_tx = dict(
>>>             txid=tx['txid'], coin=symbol, tx_timestamp=parse(tx[
↳ 'timestamp']),
>>>             from_account=tx['from'], to_account=tx['to'], memo=tx[
↳ 'memo'],
>>>             amount=Decimal(tx['quantity'])
>>>         )
>>>         yield clean_tx
>>>     except:
>>>         log.exception('Error parsing transaction data. Skipping this_
↳ TX. tx = %s', tx)
>>>         continue

```

**Parameters**

- **symbol** – The symbol of the token being filtered
- **transactions** – A list<dict> of transactions to filter
- **account** – The ‘to’ account or crypto address to filter by (only required for account-based loaders)

**Returns** A generator yielding dict’s conforming to `models.Deposit`, check the PyDoc return info for `coin_handlers.base.BaseLoader.list_txs()` for current format.

**list\_txs** (*batch=100*) → Generator[dict, None, None]

Yield transactions for all coins in *self.coins* as a generator, loads transactions in batches of *batch* and returns them seamlessly using a generator.

If *load()* hasn’t been ran already, it will automatically call *self.load()*

**Parameters** **batch** – Amount of transactions to load per batch

**Return** Generator[dict, None, None] Generator yielding dict’s that conform to `models.Deposit`

**load** (*tx\_count=1000*)

Simply imports *tx\_count* into an instance variable, and then sets *self.loaded* to True.

If *self.need\_account* is set to True by a child/parent class, this method will remove any coins from *self.coins* and *self.symbols* which have a blank/null *our\_account* in the DB, ensuring that you can trust that all coins listed in symbols/coins have an *our\_account* which isn’t empty or None.

**Parameters** **tx\_count** (*int*) – The amount of transactions to load per symbol specified in constructor

**load\_batch** (*symbol, limit=100, offset=0, account=None*)

This function should load *limit* transactions in their raw format from your data source, skipping the *offset* newest TXs efficiently, and store them in the instance var *self.transactions*

If you use the included decorator `decorators.retry_on_err()`, if any exceptions are thrown by your method, it will simply re-run it with the same arguments up to 3 tries by default.

Basic implementation:

```
>>> @retry_on_err()
>>> def load_batch(self, symbol, limit=100, offset=0, account=None):
>>>     self.transactions = self.my_rpc.get_tx_list(limit, offset)
```

#### Parameters

- **symbol** – The symbol to load a batch of transactions for
- **limit** – The amount of transactions to load
- **offset** – Skip this many transactions (most recent first)
- **account** – An account name, or coin address to filter transactions using

#### 3.2.1.1.4.5 Base Decorators

payments.coin\_handlers.base.decorators.**retry\_on\_err** (*max\_retries: int = 3, delay: int = 3, \*\*retry\_conf*)

Decorates a function or class method, wraps the function/method with a try/catch block, and will automatically re-run the function with the same arguments up to *max\_retries* time after any exception is raised, with a delay second delay between re-tries.

If it still throws an exception after *max\_retries* retries, it will log the exception details with *fail\_msg*, and then re-raise it.

Usage (retry up to 5 times, 1 second between retries, stop immediately if IOError is detected):

```
>>> @retry_on_err(5, 1, fail_on=[IOError])
... def my_func(self, some=None, args=None):
...     if some == 'io': raise IOError()
...     raise FileExistsError()
```

This will be re-ran 5 times, 1 second apart after each exception is raised, before giving up:

```
>>> my_func()
```

Where-as this one will immediately re-raise the caught IOError on the first attempt, as it's passed in *fail\_on*:

```
>>> my_func('io')
```

#### Parameters

- **max\_retries** (*int*) – Maximum total retry attempts before giving up
- **delay** (*int*) – Amount of time in seconds to sleep before re-trying the wrapped function
- **retry\_conf** – Less frequently used arguments, pass in as keyword args:
- (list) *fail\_on*: A list() of Exception types that should result in immediate failure (don't retry, raise)
- (str) *retry\_msg*: Override the log message used for retry attempts. First message param %s is func name, second message param %d is retry attempts remaining
- (str) *fail\_msg*: Override the log message used after all retry attempts are exhausted. First message param %s is func name, and second param %d is amount of times retried.

#### 3.2.1.1.4.6 Base Exceptions

**exception** `payments.coin_handlers.base.exceptions.AccountNotFound`

Bases: `payments.coin_handlers.base.exceptions.CoinHandlerException`

The sending or receiving account requested doesn't exist

**exception** `payments.coin_handlers.base.exceptions.AuthorityMissing`

Bases: `payments.coin_handlers.base.exceptions.CoinHandlerException`

Missing private key or other authorization for this operation

**exception** `payments.coin_handlers.base.exceptions.CoinHandlerException`

Bases: `Exception`

Base exception for all Coin handler exceptions to inherit

**exception** `payments.coin_handlers.base.exceptions.DeadAPIError`

Bases: `payments.coin_handlers.base.exceptions.CoinHandlerException`

A main API, e.g. a coin daemon or public node used by this coin handler is offline.

**exception** `payments.coin_handlers.base.exceptions.IssueNotSupported`

Bases: `payments.coin_handlers.base.exceptions.CoinHandlerException`

This class does not support issuing, the token name cannot be issued, or other issue problems.

**exception** `payments.coin_handlers.base.exceptions.IssuerKeyError`

Bases: `payments.coin_handlers.base.exceptions.AuthorityMissing`

Attempted to issue tokens you don't have the issuer key for

**exception** `payments.coin_handlers.base.exceptions.MissingTokenMetadata`

Bases: `payments.coin_handlers.base.exceptions.CoinHandlerException`

Could not process a transaction or run the requested Loader/Manager method as required coin metadata is missing, such as `payments.models.Coin.our_account` or a required key in the custom JSON settings.

**exception** `payments.coin_handlers.base.exceptions.NotEnoughBalance`

Bases: `payments.coin_handlers.base.exceptions.CoinHandlerException`

The sending account does not have enough balance for this operation

**exception** `payments.coin_handlers.base.exceptions.TokenNotFound`

Bases: `payments.coin_handlers.base.exceptions.CoinHandlerException`

The token/coin requested doesn't exist

#### 3.2.1.1.4.7 Module contents

#### 3.2.1.2 Module contents

This module init file is responsible for loading the Coin Handler modules, and offering methods for accessing loaders and managers.

A **Coin Handler** is a Python module (folder containing classes and init file) designed to handle sending/receiving cryptocurrency/tokens for a certain network, or certain family of networks sharing similar code.

They may handle just one single coin, several coins, or they may even allow users to dynamically add coins by querying for a specific `coin_type` from the model `payments.models.Coin`

A coin handler must contain:

- An `__init__.py` with a dictionary named `exports`, containing the keys ‘loader’ and/or ‘manager’ pointing to the un-instantiated loader/manager class.
  - If your init file needs to do some sort-of initialisation, such as dynamically generating `provides` for your classes, or adding a new coin type to `settings.COIN_TYPES`, it’s best to place it in a function named “reload” with a global boolean `loaded` so that you only initialise the module the first time it’s loaded.  
  
See the example `__init__.py` near the bottom of this module docstring.  
  
This is optional, but it will allow `reload_handlers()` to properly re-trigger your initialisation code only when changes occur, such as Coin’s being created/updated in the database.
- Two classes, a **Loader** and a **Manager**. Each class can either in it’s own file, or in a single file containing other classes / functions.
  - A **Loader** is a class which extends `base.BaseLoader`, and is responsible for retrieving transactions that occur on that coin to detect incoming transactions.
  - A **Manager** is a class which extends `base.BaseManager`, and is responsible for handling sending/issuing of coins/tokens, as well as other small functions such as validating addresses, and checking balances.

Your **Loader** class may choose to extend the helper class `base.BatchLoader`, allowing your loader to use batches/chunking for memory efficiency, without having to write much code.

Your Coin Handler classes should ONLY use the exceptions in `base.exceptions`, along with any exceptions listed in the `:raises:` pydoc statements of the overridden method.

For handling automatic retry when something goes wrong, you can use the decorator `base.decorators.retry_on_err()`

Example `__init__.py`:

```
>>> from django.conf import settings
>>> from payments.coin_handlers.SteemEngine.SteemEngineLoader import SteemEngineLoader
>>> from payments.coin_handlers.SteemEngine.SteemEngineManager import _
↪SteemEngineManager
>>>
>>> loaded = False
>>>
>>> def reload():
>>>     global loaded
>>>     if 'steemengine' not in dict(settings.COIN_TYPES):
>>>         settings.COIN_TYPES += (('steemengine', 'SteemEngine Token',),)
>>>         loaded = True
>>>
>>> if not loaded:
>>>     reload()
>>>
>>> exports = {
>>>     "loader": SteemEngineLoader,
>>>     "manager": SteemEngineManager
>>> }
```

For an example of how to layout your coin handler module, check out the pre-included Coin Handlers:

- `SteemEngine`
- `Bitcoin`

`payments.coin_handlers.add_handler(handler, handler_type)`



```
payments.coin_handlers.ch_base = 'payments.coin_handlers'
```

Base module path to where the coin handler modules are located. E.g. payments.coin\_handlers

```
payments.coin_handlers.get_loader(symbol: str) → payments.coin_handlers.base.BaseLoader.BaseLoader
```

For some use-cases, you may want to just grab the first loader that supports this coin.

```
>>> m = get_loader('ENG')
>>> m.send(amount=Decimal(1), from_address='someguy123', address='privex')
```

**Parameters** **symbol** – The coin symbol to get the loader for (uppercase)

**Return** **BaseLoader** An instance implementing *base.BaseLoader*

```
payments.coin_handlers.get_loaders(symbol: str = None) → list
```

Get all loader's, or all loader's for a certain coin

**Parameters** **symbol** – The coin symbol to get all loaders for (uppercase)

**Return** **list** If symbol not specified, a list of tuples (symbol, list<BaseLoader>.)

**Return** **list** If symbol IS specified, a list of instantiated *base.BaseLoader*'s

```
payments.coin_handlers.get_manager(symbol: str) → payments.coin_handlers.base.BaseManager.BaseManager
```

For some use-cases, you may want to just grab the first manager that supports this coin.

```
>>> m = get_manager('ENG')
>>> m.send(amount=Decimal(1), from_address='someguy123', address='privex')
```

**Parameters** **symbol** – The coin symbol to get the manager for (uppercase)

**Return** **BaseManager** An instance implementing *base.BaseManager*

```
payments.coin_handlers.get_managers(symbol: str = None) → list
```

Get all manager's, or all manager's for a certain coin

**Parameters** **symbol** – The coin symbol to get all managers for (uppercase)

**Return** **list** If symbol not specified, a list of tuples (symbol, list<BaseManager>.)

**Return** **list** If symbol IS specified, a list of instantiated *base.BaseManager*'s

```
payments.coin_handlers.handlers = {}
```

A dictionary of coin symbols, containing instantiated managers (BaseManager) and loaders (BaseLoader)

Example layout:

```
handlers = {
    'ENG': {
        'loaders': [ SteemEngineLoader, ],
        'managers': [ SteemEngineLoader, ],
    },
    'SGTK': {
        'loaders': [ SteemEngineLoader, ],
        'managers': [ SteemEngineLoader, ],
    },
}
```

```
payments.coin_handlers.handlers_loaded = False
```

Used to track whether the Coin Handlers have been initialized, so reload\_handlers can be auto-called.

`payments.coin_handlers.has_loader(symbol: str) → bool`

Helper function - does this symbol have a loader class?

`payments.coin_handlers.has_manager(symbol: str) → bool`

Helper function - does this symbol have a manager class?

`payments.coin_handlers.is_database_synchronized(database: str) → bool`

Check if all migrations have been ran. Useful for preventing auto-running code accessing models before the tables even exist, thus preventing you from migrating...

```
>>> from django.db import DEFAULT_DB_ALIAS
>>> if not is_database_synchronized(DEFAULT_DB_ALIAS):
>>>     log.warning('Cannot run reload_handlers because there are unapplied_
↳ migrations!')
>>>     return
```

**Parameters** `database` (*str*) – Which Django database config is being used? Generally just pass `django.db.DEFAULT_DB_ALIAS`

**Return** `bool` True if all migrations have been ran, False if not.

`payments.coin_handlers.reload_handlers()`

Resets *handler* to an empty dict, then loads all *settings.COIN\_HANDLER* classes into the dictionary *handlers* using *settings.COIN\_HANDLERS\_BASE* as the base module path to load from

## 3.2.2 payments package

### 3.2.2.1 Subpackages

### 3.2.2.2 Submodules

### 3.2.2.3 payments.admin module

`class payments.admin.AddCoinPairView(**kwargs)`

Bases: `django.views.generic.base.TemplateView`

Admin view for easily adding two coins + two pairs in each direction

`coin_types()`

View function to be called from template, for getting list of coin handler errors

`get(request, *args, **kwargs)`

`post(request, *args, **kwargs)`

`template_name = 'admin/add_pair.html'`

`class payments.admin.AddressAccountMapAdmin(model, admin_site)`

Bases: `django.contrib.admin.options.ModelAdmin`

`list_display = ('deposit_coin', 'deposit_address', 'destination_coin', 'destination_ad`

`list_filter = ('deposit_coin', 'destination_coin')`

`media`

`search_fields = ('deposit_address', 'destination_address')`

`class payments.admin.CoinAdmin(model, admin_site)`

Bases: `django.contrib.admin.options.ModelAdmin`

```
fieldsets = (('Unique Coin Symbol for refrencing from the API', {'fields': ('symbol',
get_fieldsets(request, obj=None)
    Hook for specifying fieldsets.

list_display = ('__str__', 'symbol', 'coin_type', 'enabled', 'our_account', 'can_issue
list_filter = ('coin_type',)

media

ordering = ('symbol',)

class payments.admin.CoinHealthView(**kwargs)
    Bases: django.views.generic.base.TemplateView

    Admin view for viewing health/status information of all coins in the system.

    Loads the coin handler manager for each coin, and uses the health() function to grab status info for the coin.

    Uses caching API to avoid constant RPC queries, and displays results as a standard admin view.

    get(request, *args, **kwargs)

    get_fails()
        View function to be called from template, for getting list of coin handler errors

    handler_dic()
        View function to be called from template. Loads and queries coin handlers for health, with caching.

    template_name = 'admin/coin_health.html'

class payments.admin.CoinPairAdmin(model, admin_site)
    Bases: django.contrib.admin.options.ModelAdmin

    list_display = ('__str__', 'from_coin', 'to_coin', 'exchange_rate')

    media

    ordering = ('from_coin', 'to_coin')

class payments.admin.ConversionAdmin(model, admin_site)
    Bases: django.contrib.admin.options.ModelAdmin

    list_display = ('from_coin', 'from_address', 'from_amount', 'to_coin', 'to_address', '
    list_filter = ('from_coin', 'to_coin')

    media

    ordering = ('-created_at',)

    search_fields = ('id', 'from_address', 'to_address', 'to_memo', 'to_txid')

class payments.admin.CustomAdmin(name='admin')
    Bases: django.contrib.admin.sites.AdminSite

    To allow for custom admin views, we override AdminSite, so we can add custom URLs, among other things.

    get_urls()

class payments.admin.DepositAdmin(model, admin_site)
    Bases: django.contrib.admin.options.ModelAdmin

    list_display = ('txid', 'status', 'coin', 'amount', 'address', 'from_account', 'to_acc
    list_filter = ('status', 'coin')

    media
```

```
        ordering = ('-tx_timestamp',)
        search_fields = ('id', 'txid', 'address', 'from_account', 'to_account', 'memo', 'refund')
class payments.admin.KeyPairAdmin(model, admin_site)
    Bases: django.contrib.admin.options.ModelAdmin
    list_display = ('network', 'public_key', 'account', 'key_type')
    media
    ordering = ('network', 'account')
payments.admin.clear_cache(request)
    Allow admins to clear the Django cache system
payments.admin.path(route, view, kwargs=None, name=None, *, Pattern=<class
                    'django.urls.resolvers.RoutePattern'>)
```

### 3.2.2.4 payments.apps module

```
class payments.apps.PaymentsConfig(app_name, app_module)
    Bases: django.apps.config.AppConfig
    name = 'payments'
```

### 3.2.2.5 payments.models module

This file contains Models, classes which define database tables, and how they relate to each other.

Models are used for both querying the database, as well as inserting new rows and updating existing ones.

Models may also contain **properties** and **functions** to help make them easier to use.

Note: The `coin_type` choices tuple, `COIN_TYPES` is located in `settings.py`, and may be dynamically altered by Coin Handlers. It does not enforce an enum on columns using it for `choices`, it's simply used for a dropdown list in the admin panel.

**Copyright:**

```
+=====+
|                © 2019 Privex Inc.                |
|                https://www.privex.io                |
+=====+
|
|                CryptoToken Converter                |
|
|                Core Developer(s):                    |
|
|                (+)  Chris (@someguy123) [Privex]      |
|
+=====+
```

```
class payments.models.AddressAccountMap(*args, **kwargs)
    Bases: django.db.models.base.Model
```

This database model maps normal Bitcoin-like addresses to a destination token, and their token account/address.

This is because deposits of coins such as Bitcoin/Litecoin do not contain any form of “memo”, so they must be manually mapped onto a destination.

This model may be used for handling deposits for both memo-based (Bitshares-like) and address-based (Bitcoin-like) deposits, as there is both a memo and address (or account) field for deposits + destination coin

**exception DoesNotExist**

Bases: `django.core.exceptions.ObjectDoesNotExist`

**exception MultipleObjectsReturned**

Bases: `django.core.exceptions.MultipleObjectsReturned`

**conversions**

**deposit\_address**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**deposit\_coin**

Accessor to the related object on the forward side of a many-to-one or one-to-one (via `ForwardOneToOneDescriptor` subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

**deposit\_coin\_id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**deposit\_memo**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**destination\_address**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**destination\_coin**

Accessor to the related object on the forward side of a many-to-one or one-to-one (via `ForwardOneToOneDescriptor` subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

**destination\_coin\_id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**destination\_memo**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**objects** = `<django.db.models.manager.Manager object>`

**class** `payments.models.Coin(*args, **kwargs)`

Bases: `django.db.models.base.Model`

The operator of the service should define all coins and tokens they would like to support using the Django Admin. The symbol is used as the primary key, so it must be unique. It will automatically be made uppercase. Native Coin Symbol (e.g. BTC)

**exception** `DoesNotExist`

Bases: `django.core.exceptions.ObjectDoesNotExist`

**exception** `MultipleObjectsReturned`

Bases: `django.core.exceptions.MultipleObjectsReturned`

**can\_issue**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**coin\_type**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**conversions\_from**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

**conversions\_to**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

**deposit\_convert**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

**deposit\_maps**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

#### **deposits**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

#### **dest\_maps**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

#### **display\_name**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

#### **enabled**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

#### **funds\_low**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

#### **last\_notified**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

#### **notify\_low\_funds**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

#### **objects = <django.db.models.manager.Manager object>**

#### **our\_account**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

#### **pairs**

### **pairs\_from**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

### **pairs\_to**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

### **save(\*args, \*\*kwargs)**

To avoid inconsistency, the symbol is automatically made uppercase

### **setting\_host**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **setting\_json**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **setting\_pass**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **setting\_port**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **setting\_user**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **settings**

Small helper property for quickly accessing the `setting_xxxx` fields, while also decoding the custom json field into a dictionary/list

**Returns** dict(host:str, port:str, user:str, password:str, json:dict/list)

### **should\_notify\_low**

Should we notify the admins that this coin's wallet balance is too low?

Used to rate limit "???coin wallet balance is too low" emails sent to admins.

Usage:



```
>>> from django.core.mail import mail_admins
>>> c = Coin.objects.get(symbol='BTC')
>>> if c.should_notify_low:
>>>     mail_admins('BTC hot wallet is low!', 'The hot wallet is low. Please_
↳ refill.')
```

**Return bool** True if we should notify the admins

**Return bool** False if we should skip this email notification for now, or notifications are disabled.

#### **symbol**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

#### **symbol\_id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**class** `payments.models.CoinPair(*args, **kwargs)`

Bases: `django.db.models.base.Model`

A coin pair defines an allowed conversion direction between two coins For example LTC (Litecoin) -> LTCP (Pegged Litecoin)

#### **exception DoesNotExist**

Bases: `django.core.exceptions.ObjectDoesNotExist`

#### **exception MultipleObjectsReturned**

Bases: `django.core.exceptions.MultipleObjectsReturned`

#### **exchange\_rate**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

#### **from\_coin**

Accessor to the related object on the forward side of a many-to-one or one-to-one (via `ForwardOneToOneDescriptor` subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

#### **from\_coin\_id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

#### **from\_coin\_symbol**

#### **id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**objects** = `<django.db.models.manager.Manager object>`

#### **to\_coin**

Accessor to the related object on the forward side of a many-to-one or one-to-one (via `ForwardOneToOneDescriptor` subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

### **to\_coin\_id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **to\_coin\_symbol**

**class** `payments.models.Conversion(*args, **kwargs)`

Bases: `django.db.models.base.Model`

Once a `models.Deposit` has been scanned, assuming it has a valid address or account/memo, the destination cryptocurrency/token will be sent to the user.

Successful conversion attempts are logged here, allowing for reference of where the coins came from, where they went, and what fees were taken.

### **exception DoesNotExist**

Bases: `django.core.exceptions.ObjectDoesNotExist`

### **exception MultipleObjectsReturned**

Bases: `django.core.exceptions.MultipleObjectsReturned`

### **created\_at**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **deposit**

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

### **deposit\_id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **ex\_fee**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **from\_address**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **from\_amount**

### **from\_coin**

The coin that we were sent

### **from\_coin\_id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**from\_coin\_symbol**

**get\_next\_by\_created\_at** (\*, field=<django.db.models.fields.DateTimeField: created\_at>, is\_next=True, \*\*kwargs)

**get\_next\_by\_updated\_at** (\*, field=<django.db.models.fields.DateTimeField: updated\_at>, is\_next=True, \*\*kwargs)

**get\_previous\_by\_created\_at** (\*, field=<django.db.models.fields.DateTimeField: created\_at>, is\_next=False, \*\*kwargs)

**get\_previous\_by\_updated\_at** (\*, field=<django.db.models.fields.DateTimeField: updated\_at>, is\_next=False, \*\*kwargs)

**id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**objects** = <django.db.models.manager.Manager object>

**to\_address**

Where was it sent to?

**to\_amount**

The amount of `to_coin` that was sent, stored as a high precision Decimal

**to\_coin**

The destination token/crypto this token will be converted to

**to\_coin\_id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**to\_coin\_symbol**

**to\_memo**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**to\_txid**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**tx\_fee**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**updated\_at**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**class** payments.models.CryptoKeyPair (\*args, \*\*kwargs)

Bases: django.db.models.base.Model

This model allows for storing key pairs (generally for cryptocurrency addresses/accounts) safely in the database.

The private key is automatically encrypted with AES-128 upon saving, ensuring it cannot be read from the admin panel, any API leaks, or third party applications reading from the database.

For this model to function correctly, you must set ENCRYPT\_KEY in `.env` by generating an encryption key using `./manage.py generate_key`

**exception** DoesNotExist

Bases: django.core.exceptions.ObjectDoesNotExist

### **exception MultipleObjectsReturned**

Bases: `django.core.exceptions.MultipleObjectsReturned`

### **account**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **balance**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **key\_type**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **network**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**objects** = `<django.db.models.manager.Manager object>`

### **private\_key**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **public\_key**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

### **save** (\*args, \*\*kwargs)

To ensure that private keys can only be entered / updated from the admin panel and not viewed, we encrypt them with AES-128 when saving.

To avoid encrypting an already encrypted key, we only encrypt the key if we're sure it's not encrypted already.

#### **Raises**

- **EncryptionError** – Something went wrong while encrypting the key
- **EncryptKeyMissing** – The key `settings.ENCRIPT_KEY` is not set or is not a valid encryption key.

### **used**

For disposable addresses, e.g. Bitcoin addresses, this field tracks whether it has been used for a deposit.

### **class** `payments.models.Deposit` (\*args, \*\*kwargs)

Bases: `django.db.models.base.Model`

A log of incoming token/crypto deposits, which will later be converted into crypto.

The primary key of a *Deposit* is the auto-generated *id* field - an auto incrementing integer.

There is a composite unique constraint on (txid, coin, vout), ensuring duplicate transactions do not get stored.

Deposits start out in state `new`, as they are processed by the conversion system they progress into either:

**'err'** - **An error occurred while converting / importing** During the import/conversion there was a serious error that could not be recovered from This should be investigated by a developer.

**‘inv’ - Invalid source/destination, user did not follow instructions correctly** The coins were sent to a non-registered address, or a memo we don’t know how to process. An admin should attempt to refund these coins to the sender.

**‘refund’ - The coins sent in this Deppsit were refunded** Info about the refund should be in the `refund_*` fields

**‘mapped’ - Deposit passed initial sanity checks, and we know the destination coin, address/account and memo.** Most deposits should only stay in this state for a few seconds, before they’re converted. If a deposit stays in this state for more than a few minutes, it generally means something is wrong with the Coin Handler, preventing it from sending the coins, e.g. low balance.

**‘conv’ - Successfully Converted** The deposited coins were successfully converted into their destination coin, and there should be a related `models.Conversion` containing the conversion details.

#### **exception DoesNotExist**

Bases: `django.core.exceptions.ObjectDoesNotExist`

#### **exception MultipleObjectsReturned**

Bases: `django.core.exceptions.MultipleObjectsReturned`

**STATUSES = (('err', 'Error Processing Transaction'), ('inv', 'Transaction is invalid'))**

#### **address**

If the deposit is from a classic Bitcoin-like cryptocurrency with addresses, then you should enter the address where the coins were deposited into, in this field.

#### **amount**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

#### **coin**

The symbol of the cryptocurrency or token that was deposited, in uppercase. e.g. LTC, LTCP, BTCP, STEEMP

#### **coin\_id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

#### **coin\_symbol**

#### **conversion**

Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

#### **convert\_dest\_address**

The destination address. Set after a deposit has been analyzed, and we know what coin it will be converted to.

#### **convert\_dest\_memo**

The destination memo. Set after a deposit has been analyzed, and we know what coin it will be converted to.

#### **convert\_to**

The destination coin. Set after a deposit has been analyzed, and we know what coin it will be converted to

**convert\_to\_id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**created\_at**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**error\_reason**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**from\_account**

If account-based coin, contains the name of the account that sent the coins

**get\_next\_by\_created\_at** (\*, field=<django.db.models.fields.DateTimeField: created\_at>, is\_next=True, \*\*kwargs)

**get\_next\_by\_updated\_at** (\*, field=<django.db.models.fields.DateTimeField: updated\_at>, is\_next=True, \*\*kwargs)

**get\_previous\_by\_created\_at** (\*, field=<django.db.models.fields.DateTimeField: created\_at>, is\_next=False, \*\*kwargs)

**get\_previous\_by\_updated\_at** (\*, field=<django.db.models.fields.DateTimeField: updated\_at>, is\_next=False, \*\*kwargs)

**get\_status\_display** (\*, field=<django.db.models.fields.CharField: status>)

**id**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**last\_convert\_attempt**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**memo**

If the coin supports memos, and they're required to identify a deposit, use this field.

**objects** = <django.db.models.manager.Manager object>

**processed\_at**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**refund\_address**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**refund\_amount**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**refund\_coin**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**refund\_memo**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**refund\_txid**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**refunded\_at**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**status**

The current status of this deposit, see `STATUSES`

**to\_account**

If account-based coin, contains the name of the account that the coins were deposited into

**tx\_timestamp**

The date/time the transaction actually occurred on the chain

**txid**

The transaction ID where the coins were received.

**updated\_at**

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

**vout**

If a transaction contains multiple deposits, for example, a Bitcoin transaction that contains several outputs (vout's) for our addresses, then each vout must have an consistent output number, i.e. one that will not change each time the blockchain transaction is compared against the database.

### 3.2.2.6 payments.serializers module

```
class payments.serializers.CoinPairSerializer (instance=None, data=<class
                                             'rest_framework.fields.empty'>,
                                             **kwargs)
```

Bases: `rest_framework.serializers.HyperlinkedModelSerializer`

**class Meta**

Bases: `object`

**fields** = ('id', 'from\_coin', 'from\_coin\_symbol', 'to\_coin', 'to\_coin\_symbol', 'exch

**model**

alias of `payments.models.CoinPair`

```
class payments.serializers.CoinSerializer (instance=None, data=<class
                                             'rest_framework.fields.empty'>, **kwargs)
```

Bases: `rest_framework.serializers.HyperlinkedModelSerializer`

**class Meta**

Bases: `object`

**fields** = ('symbol', 'display\_name', 'our\_account', 'can\_issue')

**model**

alias of `payments.models.Coin`

```
class payments.serializers.ConversionSerializer (instance=None, data=<class
                                             'rest_framework.fields.empty'>,
                                             **kwargs)
```

Bases: `rest_framework.serializers.HyperlinkedModelSerializer`

```
class Meta
    Bases: object
    exclude = ()
    model
        alias of payments.models.Conversion

class payments.serializers.DepositSerializer (instance=None, data=<class
                                                'rest_framework.fields.empty'>,
                                                **kwargs)
    Bases: rest_framework.serializers.HyperlinkedModelSerializer

class Meta
    Bases: object
    fields = ('id', 'txid', 'coin', 'coin_symbol', 'vout', 'status', 'tx_timestamp', 'a
    model
        alias of payments.models.Deposit
```

### 3.2.2.7 payments.tests module

### 3.2.2.8 payments.views module

```
class payments.views.CoinAPI (**kwargs)
    Bases: rest_framework.viewsets.ReadOnlyModelViewSet
    lookup_value_regex = '[^/]+'
    queryset
    serializer_class
        alias of payments.serializers.CoinSerializer

class payments.views.CoinPairAPI (**kwargs)
    Bases: rest_framework.viewsets.ReadOnlyModelViewSet
    filterset_fields = ('from_coin', 'to_coin')
    lookup_value_regex = '[^/]+'
    queryset
    serializer_class
        alias of payments.serializers.CoinPairSerializer

class payments.views.ConversionAPI (**kwargs)
    Bases: rest_framework.viewsets.ReadOnlyModelViewSet
    filterset_fields = ('from_coin', 'to_coin', 'from_address', 'to_address', 'deposit_fr
    pagination_class
        alias of CustomPaginator
    queryset
    serializer_class
        alias of payments.serializers.ConversionSerializer

class payments.views.ConvertAPI (**kwargs)
    Bases: rest_framework.views.APIView
    authentication_classes = (<class 'payments.views.DRFNoCSRF'>,)

```



```
    post (request: rest_framework.request.Request)

class payments.views.CustomPaginator
    Bases: rest_framework.pagination.LimitOffsetPagination

    default_limit = 100

    max_limit = 1000

class payments.views.DRFNoCSRF
    Bases: rest_framework.authentication.SessionAuthentication

    enforce_csrf (request)
        Enforce CSRF validation for session based authentication.

class payments.views.DepositAPI (**kwargs)
    Bases: rest_framework.viewsets.ReadOnlyModelViewSet

    filterset_fields = ('address', 'from_account', 'to_account', 'txid', 'memo', 'conversion_rate')
    order_by = 'created'

    pagination_class
        alias of CustomPaginator

    queryset

    serializer_class
        alias of payments.serializers.DepositSerializer

class payments.views.IndexView (**kwargs)
    Bases: django.views.generic.base.TemplateView

    template_name = 'base.html'

payments.views.api_root (self, request, *args, **kwargs)

payments.views.r_err (msg, status=500)
```

### 3.2.2.9 Module contents



---

## REST API Documentation

---

CryptoToken Converter exposes a REST API under the URL `/api` to allow any application to easily interact with the system.

It uses [Django REST Framework](#) which automatically generates a lot of the code running behind the API endpoints.

### 4.1 Endpoints

For **GET** requests, any request parameters must either be sent as either:

**Standard GET parameters** - e.g. `/api/deposits/?from_address=someguy123`

**Directly in the URL** - e.g. `/api/coins/LTC`

For **POST** requests, you may send your request data/params as a normal URL encoded form, or you may choose to send it as JSON.

**application/json - JSON Encoded Body**

```
{
  "my_param": "somevalue",
  "other.param": "other value"
}
```

**application/x-www-form-urlencoded - Standard POST body**

```
my_param=somevalue&other.param=other%20value
```

### 4.2 `/api/convert/`

Starts the conversion process between two coins.

Returns the deposit details for you to send the coins to.

**Methods:** POST (URL Encoded Form, or JSON)

**POST Parameters:**

Parameter	Type	Description
from_coin	String	Symbol of the coin to convert from
to_coin	String	Symbol of the destination coin
destination	String	The address or account on to_coin for receiving your converted coins

All parameters are required.

**Errors:**

If the JSON response `error` key is present and set to `true`, the error message will be placed in `message`, and a non-200 status code will be returned, related to the error reason.

Potential errors and their status codes:

- “An unknown error has occurred... please contact support”, 500
- “You must specify ‘from\_coin’, ‘to\_coin’, and ‘destination’”, 400
- “There is no such coin pair {} -> {}”, 404
- “The destination {} address/account ‘{}’ is not valid”, 400

Example error response:

```
POST /api/convert/

HTTP 400 Bad Request
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
  "error": true,
  "message": "You must specify 'from_coin', 'to_coin', and 'destination'"
}
```

**Return Data:**

All successful requests will include `ex_rate` (the amount of `to_coin` per `from_coin`), `pair` (details about the coin pair that you have chosen), and `destination` (where the `from_coin` will be sent to).

Depending on whether the `from_coin` is an **address based** coin, or an **account/memo based** coin, the actual deposit details will be returned differently. Address based coins will return `address`, while account based coins will return `account` and `memo`.

Below are two examples to help explain this. SGTk is “Sometoken”, a SteemEngine token, meaning it’s account+memo based. LTC is Litecoin, a classic address based cryptocurrency.

**Example 1 (address based -> account based):**

```
POST /api/convert/
from_coin=LTC&to_coin=SGTK&destination=someguy123

HTTP 200 OK
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
  "ex_rate": 100000.0,
  "destination": "someguy123",
  "pair": "LTC -> SGTK (100000.0000 SGTK per LTC)",
  "address": "MJL1E5oSqFLpdL9BswKmYonxU1Cq1WKWGL"
}
```

**Example 2 (account based -> address based):**

```
POST /api/convert/
from_coin=SGTK&to_coin=LTC&destination=MVYBriQcasb6zvtGjPflKbbWcRoKWh4sAf

HTTP 200 OK
Content-Type: application/json
{
  "ex_rate": 0.01,
  "destination": "MVYBriQcasb6zvtGjPflKbbWcRoKWh4sAf",
  "pair": "SGTK -> LTC (0.0100 LTC per SGTK)",
  "memo": "LTC MVYBriQcasb6zvtGjPflKbbWcRoKWh4sAf",
  "account": "someguy123"
}
```

## 4.3 /api/deposits/

**/api/deposits/** Returns all deposit attempts received by the system. Can be filtered using the **GET Parameters** listed below.

**/api/deposits/<id>** Returns a single deposit attempt by it's ID

**Methods:** GET

**GET Parameters:**

These parameters can be used with the plain `/api/deposits/` URL, to filter deposits based on various columns.

Note: Results from `/api/deposits/` will always be returned as a list, even if there's only one.

Parameter	Type	Description
address	String	Return deposits that were sent to this address (only for address-based coins)
txid	String	Return deposits with a matching transaction ID
from_account	String	Return deposits that were sent from this account (only for account-based coins)
to_account	String	Return deposits that were sent to this account (only for account-based coins)
memo	String	Return deposits that were sent using this memo (normally only for account-based coins)

**Return Data:****Example 1 (Plain GET request):**

```
GET /api/deposits/

HTTP 200 OK
Content-Type: application/json

[
```

(continues on next page)

(continued from previous page)

```

{
  "id": 4,
  "txid": "635dd656b3bd8c61699e6066c9b3c6e74696e195",
  "coin": "http://127.0.0.1:8000/api/coins/SGTK/",
  "vout": 0,
  "status": "conv",
  "tx_timestamp": "2019-03-20T03:46:30Z",
  "address": null,
  "from_account": "privex",
  "to_account": "someguy123",
  "amount": "1.00000000000000000000",
  "memo": "LTC LKjpPtgMbcFgbJJYwzfelZtR8x4bbs2V3o",
  "processed_at": "2019-03-20T04:31:30.643406Z",
  "convert_to": "http://127.0.0.1:8000/api/coins/LTC/"
},
{
  "id": 5,
  "txid":
  ↪ "b881d1ae8cf280184960c9c2d74bc1bd230f18f5adcd7fe695239dbf46b06c45",
  "coin": "http://127.0.0.1:8000/api/coins/LTC/",
  "vout": 0,
  "status": "conv",
  "tx_timestamp": "2019-03-20T01:34:20Z",
  "address": "MFht1FmYhsRaSCHGdqomxQpjtGtsjFHDQX",
  "from_account": null,
  "to_account": null,
  "amount": "0.10000000000000000000",
  "memo": null,
  "processed_at": "2019-03-20T04:46:53.602857Z",
  "convert_to": "http://127.0.0.1:8000/api/coins/SGTK/"
}
]

```

**Example 2 (Filtering results):**

```

GET /api/deposits/?txid=635dd656b3bd8c61699e6066c9b3c6e74696e195

HTTP 200 OK
Content-Type: application/json

[
  {
    "id": 4,
    "txid": "635dd656b3bd8c61699e6066c9b3c6e74696e195",
    "coin": "http://127.0.0.1:8000/api/coins/SGTK/",
    "vout": 0,
    "status": "conv",
    "tx_timestamp": "2019-03-20T03:46:30Z",
    "address": null,
    "from_account": "privex",
    "to_account": "someguy123",
    "amount": "1.00000000000000000000",
    "memo": "LTC LKjpPtgMbcFgbJJYwzfelZtR8x4bbs2V3o",
    "processed_at": "2019-03-20T04:31:30.643406Z",
    "convert_to": "http://127.0.0.1:8000/api/coins/LTC/"
  }
]

```

**Example 3 (ID Lookup):**

```
GET /api/deposits/4/

HTTP 200 OK
Content-Type: application/json

{
  "id": 4,
  "txid": "635dd656b3bd8c61699e6066c9b3c6e74696e195",
  "coin": "http://127.0.0.1:8000/api/coins/SGTK/",
  "vout": 0,
  "status": "conv",
  "tx_timestamp": "2019-03-20T03:46:30Z",
  "address": null,
  "from_account": "privex",
  "to_account": "someguy123",
  "amount": "1.000000000000000000",
  "memo": "LTC LKjpPtgMbcFgbJJYwzfelZtR8x4bbs2V3o",
  "processed_at": "2019-03-20T04:31:30.643406Z",
  "convert_to": "http://127.0.0.1:8000/api/coins/LTC/"
}
```

## 4.4 /api/conversions/

**/api/conversions/** Returns all successful conversions sent by the system. Can be filtered using the **GET Parameters** listed below.

**/api/conversions/<id>** Returns a single conversion by it's ID

**Methods:** GET

**GET Parameters:**

These parameters can be used with the plain `/api/conversions/` URL, to filter conversions based on various columns.

Note: Results from `/api/conversions/` will always be returned as a list, even if there's only one.

Parameter	Type	Description
to_address	String	Return conversions that were sent to this address or account (it's used for both)
to_txid	String	Return conversions with this outgoing TXID
to_coin	String	Return conversions into this coin symbol
from_coin	String	Return conversions from this coin symbol
from_address	String	Return conversions that were sent from this address or account (it's used for both)

**Return Data:**

Note: The `to_amount` is the final amount that the user should have received AFTER `ex_fee` and `tx_fee` were removed.

**Example 1 (Plain GET request):**

```
GET /api/conversions/

HTTP 200 OK
```

(continues on next page)

(continued from previous page)

```

Content-Type: application/json
[
  {
    "url": "http://127.0.0.1:8000/api/conversions/6/",
    "from_address": "LKjpPtgMbcFgbJJYwzfelZtR8x4bbs2V3o",
    "to_address": "LKjpPtgMbcFgbJJYwzfelZtR8x4bbs2V3o",
    "to_memo": "Token Conversion from SGTK account privex",
    "to_amount": "0.008832000000000000",
    "to_txid":
    ↪ "e4a5cb3ccc5524e20a39b1a076cef16a85efc68bf929e7a3ec4a834c30711e55",
    "tx_fee": "0.000168000000000000",
    "ex_fee": "0.001000000000000000",
    "created_at": "2019-03-21T10:14:20.021360Z",
    "updated_at": "2019-03-21T10:14:20.021373Z",
    "deposit": "http://127.0.0.1:8000/api/deposits/10/",
    "from_coin": "http://127.0.0.1:8000/api/coins/SGTK/",
    "to_coin": "http://127.0.0.1:8000/api/coins/LTC/"
  },
  {
    "url": "http://127.0.0.1:8000/api/conversions/7/",
    "from_address": "someguy123",
    "to_address": "privex",
    "to_memo": "Token Conversion via LTC deposit address_
    ↪ MTcPHSipXBzwhTWT8wXMTNf6vwAxovjpx9",
    "to_amount": "900.0000000000000000",
    "to_txid": "55c30e43088c8aa6d7a74dale29d3843cd7157e7",
    "tx_fee": "0.0000000000000000",
    "ex_fee": "100.0000000000000000",
    "created_at": "2019-03-21T10:15:47.071323Z",
    "updated_at": "2019-03-21T10:15:47.071340Z",
    "deposit": "http://127.0.0.1:8000/api/deposits/9/",
    "from_coin": "http://127.0.0.1:8000/api/coins/LTC/",
    "to_coin": "http://127.0.0.1:8000/api/coins/SGTK/"
  }
]

```

**Example 2 (Filtering results):**

```

GET /api/conversions/?from_coin=SGTK&to_coin=LTC

HTTP 200 OK
Content-Type: application/json
[
  {
    "url": "http://127.0.0.1:8000/api/conversions/6/",
    "from_address": "LKjpPtgMbcFgbJJYwzfelZtR8x4bbs2V3o",
    "to_address": "LKjpPtgMbcFgbJJYwzfelZtR8x4bbs2V3o",
    "to_memo": "Token Conversion from SGTK account privex",
    "to_amount": "0.008832000000000000",
    "to_txid":
    ↪ "e4a5cb3ccc5524e20a39b1a076cef16a85efc68bf929e7a3ec4a834c30711e55",
    "tx_fee": "0.000168000000000000",
    "ex_fee": "0.001000000000000000",
    "created_at": "2019-03-21T10:14:20.021360Z",
    "updated_at": "2019-03-21T10:14:20.021373Z",
    "deposit": "http://127.0.0.1:8000/api/deposits/10/",
    "from_coin": "http://127.0.0.1:8000/api/coins/SGTK/",

```

(continues on next page)



(continued from previous page)

```

    "to_coin": "http://127.0.0.1:8000/api/coins/LTC/"
  },
  {
    "url": "http://127.0.0.1:8000/api/conversions/5/",
    "from_address": "LKjpPtgMbcFgbJJYwzfelZtR8x4bbs2V3o",
    "to_address": "LKjpPtgMbcFgbJJYwzfelZtR8x4bbs2V3o",
    "to_memo": "Token Conversion from SGTK account privex",
    "to_amount": "0.00433200000000000000",
    "to_txid": null,
    "tx_fee": "0.00016800000000000000",
    "ex_fee": "0.00050000000000000000",
    "created_at": "2019-03-20T04:56:53.859675Z",
    "updated_at": "2019-03-20T04:56:53.859691Z",
    "deposit": "http://127.0.0.1:8000/api/deposits/7/",
    "from_coin": "http://127.0.0.1:8000/api/coins/SGTK/",
    "to_coin": "http://127.0.0.1:8000/api/coins/LTC/"
  }
]

```

**Example 3 (ID Lookup):**

```

GET /api/conversions/5/

HTTP 200 OK
Content-Type: application/json

{
  "url": "http://127.0.0.1:8000/api/conversions/5/",
  "from_address": "LKjpPtgMbcFgbJJYwzfelZtR8x4bbs2V3o",
  "to_address": "LKjpPtgMbcFgbJJYwzfelZtR8x4bbs2V3o",
  "to_memo": "Token Conversion from SGTK account privex",
  "to_amount": "0.00433200000000000000",
  "to_txid": null,
  "tx_fee": "0.00016800000000000000",
  "ex_fee": "0.00050000000000000000",
  "created_at": "2019-03-20T04:56:53.859675Z",
  "updated_at": "2019-03-20T04:56:53.859691Z",
  "deposit": "http://127.0.0.1:8000/api/deposits/7/",
  "from_coin": "http://127.0.0.1:8000/api/coins/SGTK/",
  "to_coin": "http://127.0.0.1:8000/api/coins/LTC/"
}

```

## 4.5 /api/pairs/

**/api/pairs/** Returns all coin pairs supported by the system Can be filtered using the **GET Parameters** listed below.

**/api/pairs/<id>** Returns a single coin pair by it's ID

**Methods:** GET

**GET Parameters:**

These parameters can be used with the plain /api/pairs/ URL, to filter coin pairs based on from/to symbol.

Note: Results from /api/pairs/ will always be returned as a list, even if there's only one.

Parameter	Type	Description
to_coin	String	Return pairs with this destination coin symbol
from_coin	String	Return pairs with this deposit coin symbol

### Example 1 (Plain GET request):

```
GET /api/pairs/

HTTP 200 OK
Content-Type: application/json

[
  {
    "id": 1,
    "from_coin": "http://127.0.0.1:8000/api/coins/LTC/",
    "from_coin_symbol": "LTC",
    "to_coin": "http://127.0.0.1:8000/api/coins/SGTK/",
    "to_coin_symbol": "SGTK",
    "exchange_rate": "100000.00000000000000000000",
    "__str__": "LTC -> SGTK (100000.0000 SGTK per LTC) "
  },
  {
    "id": 2,
    "from_coin": "http://127.0.0.1:8000/api/coins/SGTK/",
    "from_coin_symbol": "SGTK",
    "to_coin": "http://127.0.0.1:8000/api/coins/LTC/",
    "to_coin_symbol": "LTC",
    "exchange_rate": "0.01000000000000000000",
    "__str__": "SGTK -> LTC (0.0100 LTC per SGTK) "
  }
]
```

### Example 2 (Filtering results):

```
GET /api/pairs/?from_coin=LTC

HTTP 200 OK
Content-Type: application/json

[
  {
    "id": 1,
    "from_coin": "http://127.0.0.1:8000/api/coins/LTC/",
    "from_coin_symbol": "LTC",
    "to_coin": "http://127.0.0.1:8000/api/coins/SGTK/",
    "to_coin_symbol": "SGTK",
    "exchange_rate": "100000.00000000000000000000",
    "__str__": "LTC -> SGTK (100000.0000 SGTK per LTC) "
  }
]
```

### Example 3 (ID Lookup):

```
GET /api/pairs/1/

HTTP 200 OK
```

(continues on next page)

(continued from previous page)

Content-Type: application/json

```
{
  "id": 1,
  "from_coin": "http://127.0.0.1:8000/api/coins/LTC/",
  "from_coin_symbol": "LTC",
  "to_coin": "http://127.0.0.1:8000/api/coins/SGTK/",
  "to_coin_symbol": "SGTK",
  "exchange_rate": "100000.00000000000000000000",
  "__str__": "LTC -> SGTK (100000.0000 SGTK per LTC)"
}
```



## CHAPTER 5

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### p

payments, 61  
payments.admin, 46  
payments.apps, 48  
payments.coin\_handlers, 43  
payments.coin\_handlers.base, 43  
payments.coin\_handlers.base.BaseLoader, 34  
payments.coin\_handlers.base.BaseManager, 35  
payments.coin\_handlers.base.BatchLoader, 39  
payments.coin\_handlers.base.decorators, 42  
payments.coin\_handlers.base.exceptions, 43  
payments.coin\_handlers.Bitcoin, 17  
payments.coin\_handlers.Bitcoin.BitcoinLoader, 18  
payments.coin\_handlers.Bitcoin.BitcoinManager, 20  
payments.coin\_handlers.Bitcoin.BitcoinMixin, 22  
payments.coin\_handlers.Steem, 28  
payments.coin\_handlers.Steem.SteemLoader, 29  
payments.coin\_handlers.Steem.SteemManager, 31  
payments.coin\_handlers.SteemEngine, 23  
payments.coin\_handlers.SteemEngine.SteemEngineLoader, 23  
payments.coin\_handlers.SteemEngine.SteemEngineManager, 24  
payments.models, 48  
payments.serializers, 59  
payments.tests, 60  
payments.views, 60  
steemengine.helpers, 14  
steemengine.settings, 14  
steemengine.settings.core, 11  
steemengine.settings.custom, 12  
steemengine.settings.log, 13  
steemengine.urls, 16  
steemengine.wsgi, 16

### S

steemengine, 17





## A

account (*payments.models.CryptoKeyPair* attribute), 56  
 AccountNotFound, 43  
 add\_handler() (in module *payments.coin\_handlers*), 44  
 AddCoinPairView (class in *payments.admin*), 46  
 address (*payments.models.Deposit* attribute), 57  
 address\_valid() (in module *payments.coin\_handlers*), 36  
 address\_valid() (in module *payments.coin\_handlers.Bitcoin.BitcoinManager.BitcoinManager*), 20  
 address\_valid() (in module *payments.coin\_handlers.Steem.SteemManager.SteemManager*), 32  
 address\_valid() (in module *payments.coin\_handlers.SteemEngine.SteemEngineManager.SteemEngineManager*), 25  
 AddressAccountMap (class in *payments.models*), 48  
 AddressAccountMap.DoesNotExist, 49  
 AddressAccountMap.MultipleObjectsReturned, 49  
 AddressAccountMapAdmin (class in *payments.admin*), 46  
 all\_coins (*payments.coin\_handlers.Bitcoin.BitcoinManager.BitcoinManager* attribute), 22  
 amount (*payments.models.Deposit* attribute), 57  
 api\_root() (in module *payments.views*), 61  
 authentication\_classes (in module *payments.views.ConvertAPI* attribute), 60  
 AuthorityMissing, 43

## B

balance (*payments.models.CryptoKeyPair* attribute), 56  
 balance() (*payments.coin\_handlers.base.BaseManager.BaseManager* method), 36

balance() (*payments.coin\_handlers.Bitcoin.BitcoinManager.BitcoinManager* method), 20  
 balance() (*payments.coin\_handlers.Steem.SteemManager.SteemManager* method), 32  
 balance() (*payments.coin\_handlers.SteemEngine.SteemEngineManager.SteemEngineManager* method), 25  
 BaseLoader (class in *payments.coin\_handlers.base.BaseLoader*), 34  
 BaseManager (class in *payments.coin\_handlers.base.BaseManager*), 35  
 BatchLoader (class in *payments.coin\_handlers.base.BatchLoader*), 39  
 BitcoinLoader (class in *payments.coin\_handlers.Bitcoin.BitcoinLoader*), 18  
 BitcoinManager (class in *payments.coin\_handlers.Bitcoin.BitcoinManager*), 20  
 BitcoinMixin (class in *payments.coin\_handlers.Bitcoin.BitcoinMixin*), 22

## C

can\_issue (*payments.models.Coin* attribute), 50  
 ch\_base (in module *payments.coin\_handlers*), 44  
 clean\_tx() (*payments.coin\_handlers.Steem.SteemLoader.SteemLoader* static method), 30  
 clean\_txs() (*payments.coin\_handlers.base.BatchLoader.BatchLoader* method), 40  
 clean\_txs() (*payments.coin\_handlers.Bitcoin.BitcoinLoader.BitcoinLoader* method), 19  
 clean\_txs() (*payments.coin\_handlers.Steem.SteemLoader.SteemLoader* method), 30  
 clean\_txs() (*payments.coin\_handlers.SteemEngine.SteemEngineLoader.SteemEngineLoader* method), 24

`clear_cache()` (in module `payments.admin`), 48  
`Coin` (class in `payments.models`), 49  
`coin` (`payments.models.Deposit` attribute), 57  
`Coin.DoesNotExist`, 50  
`Coin.MultipleObjectsReturned`, 50  
`coin_id` (`payments.models.Deposit` attribute), 57  
`coin_symbol` (`payments.models.Deposit` attribute), 57  
`coin_type` (`payments.models.Coin` attribute), 50  
`coin_types()` (`payments.admin.AddCoinPairView` method), 46  
`CoinAdmin` (class in `payments.admin`), 46  
`CoinAPI` (class in `payments.views`), 60  
`CoinHandlerException`, 43  
`CoinHealthView` (class in `payments.admin`), 47  
`CoinPair` (class in `payments.models`), 53  
`CoinPair.DoesNotExist`, 53  
`CoinPair.MultipleObjectsReturned`, 53  
`CoinPairAdmin` (class in `payments.admin`), 47  
`CoinPairAPI` (class in `payments.views`), 60  
`CoinPairSerializer` (class in `payments.serializers`), 59  
`CoinPairSerializer.Meta` (class in `payments.serializers`), 59  
`CoinSerializer` (class in `payments.serializers`), 59  
`CoinSerializer.Meta` (class in `payments.serializers`), 59  
`config_logger()` (in module `steemengine.settings.log`), 13  
`Conversion` (class in `payments.models`), 54  
`conversion` (`payments.models.Deposit` attribute), 57  
`Conversion.DoesNotExist`, 54  
`Conversion.MultipleObjectsReturned`, 54  
`ConversionAdmin` (class in `payments.admin`), 47  
`ConversionAPI` (class in `payments.views`), 60  
`conversions` (`payments.models.AddressAccountMap` attribute), 49  
`conversions_from` (`payments.models.Coin` attribute), 50  
`conversions_to` (`payments.models.Coin` attribute), 50  
`ConversionSerializer` (class in `payments.serializers`), 59  
`ConversionSerializer.Meta` (class in `payments.serializers`), 59  
`convert_dest_address` (`payments.models.Deposit` attribute), 57  
`convert_dest_memo` (`payments.models.Deposit` attribute), 57  
`convert_to` (`payments.models.Deposit` attribute), 57  
`convert_to_id` (`payments.models.Deposit` attribute), 57  
`ConvertAPI` (class in `payments.views`), 60  
`created_at` (`payments.models.Conversion` attribute), 54

`created_at` (`payments.models.Deposit` attribute), 58  
`CryptoKeyPair` (class in `payments.models`), 55  
`CryptoKeyPair.DoesNotExist`, 55  
`CryptoKeyPair.MultipleObjectsReturned`, 55

`CustomAdmin` (class in `payments.admin`), 47  
`CustomPaginator` (class in `payments.views`), 61

## D

`DeadAPIError`, 43  
`decrypt_str()` (in module `steemengine.helpers`), 14  
`default_limit` (`payments.views.CustomPaginator` attribute), 61  
`Deposit` (class in `payments.models`), 56  
`deposit` (`payments.models.Conversion` attribute), 54  
`Deposit.DoesNotExist`, 57  
`Deposit.MultipleObjectsReturned`, 57  
`deposit_address` (`payments.models.AddressAccountMap` attribute), 49  
`deposit_coin` (`payments.models.AddressAccountMap` attribute), 49  
`deposit_coin_id` (`payments.models.AddressAccountMap` attribute), 49  
`deposit_convert` (`payments.models.Coin` attribute), 50  
`deposit_id` (`payments.models.Conversion` attribute), 54  
`deposit_maps` (`payments.models.Coin` attribute), 50  
`deposit_memo` (`payments.models.AddressAccountMap` attribute), 49  
`DepositAdmin` (class in `payments.admin`), 47  
`DepositAPI` (class in `payments.views`), 61  
`deposits` (`payments.models.Coin` attribute), 51  
`DepositSerializer` (class in `payments.serializers`), 60  
`DepositSerializer.Meta` (class in `payments.serializers`), 60  
`dest_maps` (`payments.models.Coin` attribute), 51  
`destination_address` (`payments.models.AddressAccountMap` attribute), 49  
`destination_coin` (`payments.models.AddressAccountMap` attribute), 49  
`destination_coin_id` (`payments.models.AddressAccountMap` attribute), 49  
`destination_memo` (`payments.models.AddressAccountMap` attribute), 49

`display_name` (*payments.models.Coin attribute*), 51  
`DRFNoCSRF` (*class in payments.views*), 61

## E

`empty()` (*in module steemengine.helpers*), 15  
`enabled` (*payments.models.Coin attribute*), 51  
`encrypt_str()` (*in module steemengine.helpers*), 15  
`enforce_csrf()` (*payments.views.DRFNoCSRF method*), 61  
`error_reason` (*payments.models.Deposit attribute*), 58  
`ex_fee` (*payments.models.Conversion attribute*), 54  
`exchange_rate` (*payments.models.CoinPair attribute*), 53  
`exclude` (*payments.serializers.ConversionSerializer.Meta attribute*), 60

## F

`fields` (*payments.serializers.CoinPairSerializer.Meta attribute*), 59  
`fields` (*payments.serializers.CoinSerializer.Meta attribute*), 59  
`fields` (*payments.serializers.DepositSerializer.Meta attribute*), 60  
`fieldsets` (*payments.admin.CoinAdmin attribute*), 46  
`filterset_fields` (*payments.views.CoinPairAPI attribute*), 60  
`filterset_fields` (*payments.views.ConversionAPI attribute*), 60  
`filterset_fields` (*payments.views.DepositAPI attribute*), 61  
`from_account` (*payments.models.Deposit attribute*), 58  
`from_address` (*payments.models.Conversion attribute*), 54  
`from_amount` (*payments.models.Conversion attribute*), 54  
`from_coin` (*payments.models.CoinPair attribute*), 53  
`from_coin` (*payments.models.Conversion attribute*), 54  
`from_coin_id` (*payments.models.CoinPair attribute*), 53  
`from_coin_id` (*payments.models.Conversion attribute*), 54  
`from_coin_symbol` (*payments.models.CoinPair attribute*), 53  
`from_coin_symbol` (*payments.models.Conversion attribute*), 54  
`funds_low` (*payments.models.Coin attribute*), 51

## G

`get()` (*payments.admin.AddCoinPairView method*), 46  
`get()` (*payments.admin.CoinHealthView method*), 47

`get_deposit()` (*payments.coin\_handlers.base.BaseManager.BaseManager method*), 36  
`get_deposit()` (*payments.coin\_handlers.Bitcoin.BitcoinManager.BitcoinManager method*), 21  
`get_deposit()` (*payments.coin\_handlers.Steem.SteemManager.SteemManager method*), 33  
`get_deposit()` (*payments.coin\_handlers.SteemEngine.SteemEngineManager.SteemEngineManager method*), 25  
`get_fails()` (*payments.admin.CoinHealthView method*), 47  
`get_fernet()` (*in module steemengine.helpers*), 15  
`get_fieldsets()` (*payments.admin.CoinAdmin method*), 47  
`get_loader()` (*in module payments.coin\_handlers*), 45  
`get_loaders()` (*in module payments.coin\_handlers*), 45  
`get_manager()` (*in module payments.coin\_handlers*), 45  
`get_managers()` (*in module payments.coin\_handlers*), 45  
`get_next_by_created_at()` (*payments.models.Conversion method*), 55  
`get_next_by_created_at()` (*payments.models.Deposit method*), 58  
`get_next_by_updated_at()` (*payments.models.Conversion method*), 55  
`get_next_by_updated_at()` (*payments.models.Deposit method*), 58  
`get_previous_by_created_at()` (*payments.models.Conversion method*), 55  
`get_previous_by_created_at()` (*payments.models.Deposit method*), 58  
`get_previous_by_updated_at()` (*payments.models.Conversion method*), 55  
`get_previous_by_updated_at()` (*payments.models.Deposit method*), 58  
`get_status_display()` (*payments.models.Deposit method*), 58  
`get_urls()` (*payments.admin.CustomAdmin method*), 47

## H

`handler_dic()` (*payments.admin.CoinHealthView method*), 47  
`handlers` (*in module payments.coin\_handlers*), 45  
`handlers_loaded` (*in module payments.coin\_handlers*), 45  
`has_loader()` (*in module payments.coin\_handlers*), 45

[has\\_manager\(\)](#) (in module `payments.coin_handlers`), [list\\_display](#) (`payments.admin.CoinPairAdmin` attribute), [46](#), [47](#)  
[health\(\)](#) (`payments.coin_handlers.base.BaseManager.BaseManager` method), [36](#), [47](#)  
[health\(\)](#) (`payments.coin_handlers.Bitcoin.BitcoinManager.BitcoinManager` method), [21](#), [47](#) (`payments.admin.DepositAdmin` attribute), [47](#)  
[health\(\)](#) (`payments.coin_handlers.Steem.SteemManager.SteemManager` method), [33](#), [48](#) (`payments.admin.KeyPairAdmin` attribute), [48](#)  
[health\(\)](#) (`payments.coin_handlers.SteemEngine.SteemEngineManager.SteemEngineManager` method), [26](#), [46](#) (`payments.admin.AddressAccountMapAdmin` attribute), [46](#)  
[health\\_test\(\)](#) (`payments.coin_handlers.base.BaseManager.BaseManager` method), [37](#), [47](#) (`payments.admin.CoinAdmin` attribute), [47](#)  
[health\\_test\(\)](#) (`payments.coin_handlers.Bitcoin.BitcoinManager.BitcoinManager` method), [21](#), [47](#) (`payments.admin.DepositAdmin` attribute), [47](#)  
[health\\_test\(\)](#) (`payments.coin_handlers.Steem.SteemManager.SteemManager` method), [33](#), [41](#) (`payments.admin.KeyPairAdmin` attribute), [41](#)  
[health\\_test\(\)](#) (`payments.coin_handlers.SteemEngine.SteemEngineManager.SteemEngineManager` method), [26](#), [30](#) (`payments.admin.AddressAccountMapAdmin` attribute), [30](#)  
[id](#) (`payments.models.AddressAccountMap` attribute), [49](#), [24](#) (`payments.coin_handlers.SteemEngine.SteemEngineLoader.SteemEngineLoader` method), [24](#)  
[id](#) (`payments.models.CoinPair` attribute), [53](#), [35](#) (`payments.coin_handlers.base.BaseLoader.BaseLoader` method), [35](#)  
[id](#) (`payments.models.Conversion` attribute), [55](#), [41](#) (`payments.coin_handlers.base.BatchLoader.BatchLoader` method), [41](#)  
[id](#) (`payments.models.CryptoKeyPair` attribute), [56](#), [31](#) (`payments.coin_handlers.Steem.SteemLoader.SteemLoader` method), [31](#)  
[id](#) (`payments.models.Deposit` attribute), [58](#), [24](#) (`payments.coin_handlers.SteemEngine.SteemEngineLoader.SteemEngineLoader` method), [24](#)  
[IndexView](#) (class in `payments.views`), [61](#), [60](#) (`payments.views.CoinAPI` attribute), [60](#)  
[is\\_database\\_synchronized\(\)](#) (in module `payments.coin_handlers`), [46](#), [60](#) (`payments.views.CoinPairAPI` attribute), [60](#)  
[is\\_encrypted\(\)](#) (in module `steemengine.helpers`), [15](#), [24](#) (`payments.coin_handlers.SteemEngine.SteemEngineLoader.SteemEngineLoader` method), [24](#)  
[issue\(\)](#) (`payments.coin_handlers.base.BaseManager.BaseManager` method), [37](#), [41](#) (`payments.coin_handlers.base.BatchLoader.BatchLoader` method), [41](#)  
[issue\(\)](#) (`payments.coin_handlers.SteemEngine.SteemEngineManager.SteemEngineManager` method), [26](#), [19](#) (`payments.coin_handlers.Bitcoin.BitcoinLoader.BitcoinLoader` method), [19](#)  
[IssueNotSupported](#), [43](#), [24](#) (`payments.coin_handlers.SteemEngine.SteemEngineLoader.SteemEngineLoader` method), [24](#)  
[IssuerKeyError](#), [43](#), [24](#) (`payments.coin_handlers.SteemEngine.SteemEngineLoader.SteemEngineLoader` method), [24](#)  
**K**  
[key\\_type](#) (`payments.models.CryptoKeyPair` attribute), [56](#), [60](#) (`payments.views.CoinAPI` attribute), [60](#)  
[KeyPairAdmin](#) (class in `payments.admin`), [48](#), [60](#) (`payments.views.CoinPairAPI` attribute), [60](#)  
**L**  
[last\\_convert\\_attempt](#) (`payments.models.Deposit` attribute), [58](#), [61](#) (`payments.views.CustomPaginator` attribute), [61](#)  
[last\\_notified](#) (`payments.models.Coin` attribute), [51](#), [46](#) (`payments.admin.AddressAccountMapAdmin` attribute), [46](#)  
[list\\_display](#) (`payments.admin.AddressAccountMapAdmin` attribute), [46](#), [47](#) (`payments.admin.CoinAdmin` attribute), [47](#)  
[list\\_display](#) (`payments.admin.CoinAdmin` attribute), [47](#), [47](#) (`payments.admin.CoinPairAdmin` attribute), [47](#)  
[list\\_display](#) (`payments.admin.ConversionAdmin` attribute), [47](#), [47](#) (`payments.admin.ConversionAdmin` attribute), [47](#)  
**M**  
[max\\_limit](#) (`payments.views.CustomPaginator` attribute), [61](#), [46](#) (`payments.admin.AddressAccountMapAdmin` attribute), [46](#)  
[media](#) (`payments.admin.AddressAccountMapAdmin` attribute), [46](#), [47](#) (`payments.admin.CoinAdmin` attribute), [47](#)  
[media](#) (`payments.admin.CoinAdmin` attribute), [47](#), [47](#) (`payments.admin.CoinPairAdmin` attribute), [47](#)  
[media](#) (`payments.admin.CoinPairAdmin` attribute), [47](#), [47](#) (`payments.admin.ConversionAdmin` attribute), [47](#)  
[media](#) (`payments.admin.ConversionAdmin` attribute), [47](#), [47](#) (`payments.admin.ConversionAdmin` attribute), [47](#)

media (*payments.admin.DepositAdmin* attribute), 47  
 media (*payments.admin.KeyPairAdmin* attribute), 48  
 memo (*payments.models.Deposit* attribute), 58  
 MissingTokenMetadata, 43  
 model (*payments.serializers.CoinPairSerializer.Meta* attribute), 59  
 model (*payments.serializers.CoinSerializer.Meta* attribute), 59  
 model (*payments.serializers.ConversionSerializer.Meta* attribute), 60  
 model (*payments.serializers.DepositSerializer.Meta* attribute), 60

## N

name (*payments.apps.PaymentsConfig* attribute), 48  
 network (*payments.models.CryptoKeyPair* attribute), 56  
 NotEnoughBalance, 43  
 notify\_low\_funds (*payments.models.Coin* attribute), 51

## O

objects (*payments.models.AddressAccountMap* attribute), 49  
 objects (*payments.models.Coin* attribute), 51  
 objects (*payments.models.CoinPair* attribute), 53  
 objects (*payments.models.Conversion* attribute), 55  
 objects (*payments.models.CryptoKeyPair* attribute), 56  
 objects (*payments.models.Deposit* attribute), 58  
 order\_by (*payments.views.DepositAPI* attribute), 61  
 ordering (*payments.admin.CoinAdmin* attribute), 47  
 ordering (*payments.admin.CoinPairAdmin* attribute), 47  
 ordering (*payments.admin.ConversionAdmin* attribute), 47  
 ordering (*payments.admin.DepositAdmin* attribute), 47  
 ordering (*payments.admin.KeyPairAdmin* attribute), 48  
 orig\_symbol (*payments.coin\_handlers.base.BaseManager.BaseManager* attribute), 37  
 our\_account (*payments.models.Coin* attribute), 51

## P

pagination\_class (*payments.views.ConversionAPI* attribute), 60  
 pagination\_class (*payments.views.DepositAPI* attribute), 61  
 pairs (*payments.models.Coin* attribute), 51  
 pairs\_from (*payments.models.Coin* attribute), 52  
 pairs\_to (*payments.models.Coin* attribute), 52  
 path() (in module *payments.admin*), 48  
 path() (in module *steemengine.urls*), 16

payments (*module*), 61  
 payments.admin (*module*), 46  
 payments.apps (*module*), 48  
 payments.coin\_handlers (*module*), 43  
 payments.coin\_handlers.base (*module*), 43  
 payments.coin\_handlers.base.BaseLoader (*module*), 34  
 payments.coin\_handlers.base.BaseManager (*module*), 35  
 payments.coin\_handlers.base.BatchLoader (*module*), 39  
 payments.coin\_handlers.base.decorators (*module*), 42  
 payments.coin\_handlers.base.exceptions (*module*), 43  
 payments.coin\_handlers.Bitcoin (*module*), 17  
 payments.coin\_handlers.Bitcoin.BitcoinLoader (*module*), 18  
 payments.coin\_handlers.Bitcoin.BitcoinManager (*module*), 20  
 payments.coin\_handlers.Bitcoin.BitcoinMixin (*module*), 22  
 payments.coin\_handlers.Steem (*module*), 28  
 payments.coin\_handlers.Steem.SteemLoader (*module*), 29  
 payments.coin\_handlers.Steem.SteemManager (*module*), 31  
 payments.coin\_handlers.SteemEngine (*module*), 23  
 payments.coin\_handlers.SteemEngine.SteemEngineLoader (*module*), 23  
 payments.coin\_handlers.SteemEngine.SteemEngineManager (*module*), 24  
 payments.models (*module*), 48  
 payments.serializers (*module*), 59  
 payments.tests (*module*), 60  
 payments.views (*module*), 60  
 PaymentsConfig (*class in payments.apps*), 48  
 post() (*payments.admin.AddCoinPairView* method), 46  
 post() (*payments.views.ConvertAPI* method), 60  
 private\_key (*payments.models.CryptoKeyPair* attribute), 56  
 processed\_at (*payments.models.Deposit* attribute), 58  
 provides (*payments.coin\_handlers.base.BaseLoader.BaseLoader* attribute), 35  
 provides (*payments.coin\_handlers.base.BaseManager.BaseManager* attribute), 37  
 provides (*payments.coin\_handlers.Bitcoin.BitcoinLoader.BitcoinLoader* attribute), 19  
 provides (*payments.coin\_handlers.Bitcoin.BitcoinManager.BitcoinManager* attribute), 21



provides (payments.coin\_handlers.Steem.SteemLoader.SteemLoader attribute), 31

provides (payments.coin\_handlers.Steem.SteemManager.SteemManager attribute), 33

provides (payments.coin\_handlers.SteemEngine.SteemEngineLoader.SteemEngineLoader attribute), 24

provides (payments.coin\_handlers.SteemEngine.SteemEngineManager.SteemEngineManager attribute), 26

public\_key (payments.models.CryptoKeyPair attribute), 56

## Q

queryset (payments.views.CoinAPI attribute), 60

queryset (payments.views.CoinPairAPI attribute), 60

queryset (payments.views.ConversionAPI attribute), 60

queryset (payments.views.DepositAPI attribute), 61

## R

r\_err() (in module payments.views), 61

random\_str() (in module steemengine.helpers), 16

re\_path() (in module steemengine.urls), 16

refund\_address (payments.models.Deposit attribute), 58

refund\_amount (payments.models.Deposit attribute), 58

refund\_coin (payments.models.Deposit attribute), 58

refund\_memo (payments.models.Deposit attribute), 58

refund\_txid (payments.models.Deposit attribute), 58

refunded\_at (payments.models.Deposit attribute), 59

reload() (in module payments.coin\_handlers.Bitcoin), 18

reload() (in module payments.coin\_handlers.Steem), 29

reload() (in module payments.coin\_handlers.SteemEngine), 23

reload\_handlers() (in module payments.coin\_handlers), 46

retry\_on\_err() (in module payments.coin\_handlers.base.decorators), 42

rpcs (payments.coin\_handlers.Bitcoin.BitcoinLoader.BitcoinLoader attribute), 19

rpcs (payments.coin\_handlers.Bitcoin.BitcoinManager.BitcoinManager attribute), 21

## S

save() (payments.models.Coin method), 52

save() (payments.models.CryptoKeyPair method), 56

search\_fields (payments.admin.AddressAccountMapAdmin attribute), 46

search\_fields (payments.admin.ConversionAdmin attribute), 47

search\_fields (payments.admin.DepositAdmin attribute), 48

search\_fields (payments.coin\_handlers.base.BaseManager.BaseManager attribute), 37

search\_fields (payments.coin\_handlers.Bitcoin.BitcoinManager.BitcoinManager attribute), 21

search\_fields (payments.coin\_handlers.Steem.SteemManager.SteemManager attribute), 33

send() (payments.coin\_handlers.SteemEngine.SteemEngineManager.SteemEngineManager attribute), 26

send\_or\_issue() (payments.coin\_handlers.base.BaseManager.BaseManager attribute), 38

send\_or\_issue() (payments.coin\_handlers.SteemEngine.SteemEngineManager.SteemEngineManager attribute), 27

serializer\_class (payments.views.CoinAPI attribute), 60

serializer\_class (payments.views.CoinPairAPI attribute), 60

serializer\_class (payments.views.ConversionAPI attribute), 60

serializer\_class (payments.views.DepositAPI attribute), 61

setting (payments.coin\_handlers.Bitcoin.BitcoinManager.BitcoinManager attribute), 22

setting\_host (payments.models.Coin attribute), 52

setting\_json (payments.models.Coin attribute), 52

setting\_pass (payments.models.Coin attribute), 52

setting\_port (payments.models.Coin attribute), 52

setting\_user (payments.models.Coin attribute), 52

settings (payments.coin\_handlers.Bitcoin.BitcoinLoader.BitcoinLoader attribute), 20

settings (payments.coin\_handlers.Bitcoin.BitcoinManager.BitcoinManager attribute), 22

settings (payments.coin\_handlers.Steem.SteemLoader.SteemLoader attribute), 31

settings (payments.models.Coin attribute), 52

should\_notify\_low (payments.models.Coin attribute), 52

status (payments.models.Deposit attribute), 59

STATUSES (payments.models.Deposit attribute), 57

steemengine (module), 17

steemengine.helpers (module), 14

steemengine.settings (module), 14

steemengine.settings.core (module), 11

steemengine.settings.custom (module), 12

steemengine.settings.log (module), 13

steemengine.urls (module), 16

steemengine.wsgi (module), 16

SteemEngineLoader (class in payments.coin\_handlers.SteemEngine.SteemEngineLoader), 23

SteemEngineManager (class in pay-

[payments.coin\\_handlers.SteemEngine.SteemEngineManager](#)),  
[25](#)  
[SteemLoader](#) (class in [payments.coin\\_handlers.Steem.SteemLoader](#)),  
[29](#)  
[SteemManager](#) (class in [payments.coin\\_handlers.Steem.SteemManager](#)),  
[32](#)  
[symbol](#) ([payments.coin\\_handlers.base.BaseManager.BaseManager](#)  
[attribute](#)), [39](#)  
[symbol](#) ([payments.models.Coin](#) attribute), [53](#)  
[symbol\\_id](#) ([payments.models.Coin](#) attribute), [53](#)

## T

[template\\_name](#) ([payments.admin.AddCoinPairView](#)  
[attribute](#)), [46](#)  
[template\\_name](#) ([payments.admin.CoinHealthView](#) at-  
[tribute](#)), [47](#)  
[template\\_name](#) ([payments.views.IndexView](#) at-  
[tribute](#)), [61](#)  
[to\\_account](#) ([payments.models.Deposit](#) attribute), [59](#)  
[to\\_address](#) ([payments.models.Conversion](#) attribute),  
[55](#)  
[to\\_amount](#) ([payments.models.Conversion](#) attribute),  
[55](#)  
[to\\_coin](#) ([payments.models.CoinPair](#) attribute), [53](#)  
[to\\_coin](#) ([payments.models.Conversion](#) attribute), [55](#)  
[to\\_coin\\_id](#) ([payments.models.CoinPair](#) attribute), [54](#)  
[to\\_coin\\_id](#) ([payments.models.Conversion](#) attribute),  
[55](#)  
[to\\_coin\\_symbol](#) ([payments.models.CoinPair](#) at-  
[tribute](#)), [54](#)  
[to\\_coin\\_symbol](#) ([payments.models.Conversion](#) at-  
[tribute](#)), [55](#)  
[to\\_memo](#) ([payments.models.Conversion](#) attribute), [55](#)  
[to\\_txid](#) ([payments.models.Conversion](#) attribute), [55](#)  
[TokenNotFound](#), [43](#)  
[tx\\_fee](#) ([payments.models.Conversion](#) attribute), [55](#)  
[tx\\_timestamp](#) ([payments.models.Deposit](#) attribute),  
[59](#)  
[txid](#) ([payments.models.Deposit](#) attribute), [59](#)

## U

[updated\\_at](#) ([payments.models.Conversion](#) attribute),  
[55](#)  
[updated\\_at](#) ([payments.models.Deposit](#) attribute), [59](#)  
[used](#) ([payments.models.CryptoKeyPair](#) attribute), [56](#)

## V

[vout](#) ([payments.models.Deposit](#) attribute), [59](#)