# py-stellar-base Documentation

***Release 0.1.7***

**Stellar Community**

**Oct 28, 2018**

# Contents

CHAPTER 1

# Quickstart

Here you'll find some basic examples on how to use the libary.

## 1.1 Installation

### 1.1.1 Via pipenv or pip (Recommended)

To install py-stellar-base, use pipenv to install the module:

```
pipenv install stellar-base
```

If you're not using pipenv, you should. Otherwise, you can install it via plain old `pip`. More on installing Python and dependencies can be found over in the Hitchhiker's Guide to Python.

### 1.1.2 Via Source Code

Please use the code on pypi whenever possible. The latest code may be unstable.

You can install it from source code via pip:

```
pip install git+git://github.com/StellarCN/py-stellar-base
```

And you can always clone the repository directly, and install it locally:

```
git clone git@github.com:StellarCN/py-stellar-base.git;
cd stellar_base;
pip install .
```

## 1.2 Quick Start

### 1.2.1 Stellar Guides

At the absolute basics, you'll want to read up on Stellar's Documentation Guides, as it contains a lot of information on the concepts used below (Transactions, Payments, Operations, KeyPairs, etc.).

### 1.2.2 Alright, let's get started!

First, you'll want to create a Stellar key pair. There are 2 methods for generating a key pair:

#### Random Key Generation

Simply use *Keypair.random* to generate the object like so:

```
from stellar_base.keypair import Keypair
kp = Keypair.random()
```

#### Deterministic generation

Or, generate the key pair is deterministically from a mnemonic string, also known as "seed phrase". This can be useful for backing up the passphrase on paper and using it later, or by making it easier to memorize than the secret seed.

First you'll need to generate a mnemonic string:

```
from stellar_base.utils import StellarMnemonic
# Here we use Chinese, but English is the default language.
sm = StellarMnemonic("chinese")
secret_phrase = sm.generate()
```

You can also use your own mnemonic string instead of a generated one. Once you've created your secret phrase, you should either write your phrase down or memorize it. You should not share your mnemonic string with anyone.

From here, we use *Keypair.deterministic* to generate the keypair from your secret phrase:

```
kp = Keypair.deterministic(m, lang='chinese')
```

You can even create multiple key pairs from the same phrase, using a different index with each call. For example:

```
kp1 = Keypair.deterministic(m, lang='chinese', index=1)
kp2 = Keypair.deterministic(m, lang='chinese', index=2)
```

From the generated *Keypair* object, you can easily access your public and private key.

```
publickey = kp.address().decode()
seed = kp.seed().decode()
```

Your master public key is also your account address. If someone needs to send you a transaction, you should share your public key with them. However, your secret seed should always remain locally on your computer, and it should never be transmitted over the internet.

If you ever forget or lose the public key, you can regenerate the key pair from the your secret seed:

```
from stellar_base.keypair import Keypair
kp = Keypair.from_seed(seed)
```

Both the public key and the secret seed can be regenerated via the secret phrase if you used on.

```
from stellar_base.keypair import Keypair
seed_phrase = '...' # the word sequence that you wrote down or memorized
kp = Keypair.deterministic(seed_phrase, lang='chinese')
```

However, if you used a random generator, it is important to never lose your seed - otherwise you won't be able to send transactions, and many other operations!

Here is a sample key pair in Stellar Development Foundation's (SDF) TESTNET; let's use them in the following steps:

```
publickey = 'GDVDKQFP665JAO7A2LSHNLQIUNYNAAIGJ6FYJVMG4DT3YJQQJSRBLQDG'
seed = 'SCVLSUGYEAUC4MVWJORB63JBMY2CEX6ATTJ5MXTENGD3IELUQF4F6HUB'
```

### 1.2.3 Create An Account

Now, in order to create an account, you need to run a *CreateAccount* operation with your new account ID. Due to Stellar's account minimums, you'll need to transfer the minimum account balance from another account with the create account operation. As of this writing, minimum balance is `1 XLM (2 x 0.5 Base Reserve)`, and is subject to change.

#### Using The SDF Testnet

If you want to play in the Stellar test network, you can ask our Friendbot to create an account for you as shown below:

```
import requests
publickey = kp.address().decode()
url = 'https://friendbot.stellar.org'
r = requests.get(url, params={'addr': publickey})
```

#### Using The Stellar Live Network

On the other hand, if you would like to create an account on the live network, you should buy some Stellar Lumens from an exchange. When you withdraw the Lumens into your new account, the exchange will automatically create the account for you. However, if you want to create an account from another account of your own, here's an example of how to do so:

```
from stellar_base.keypair import Keypair
from stellar_base.asset import Asset
from stellar_base.operation import CreateAccount, Payment
from stellar_base.transaction import Transaction
from stellar_base.transaction_envelope import TransactionEnvelope as Te
from stellar_base.memo import TextMemo
from stellar_base.horizon import horizon_livenet

# This creates a new Horizon Livenet instance
horizon = horizon_livenet()

# This is the seed (the StrKey representation of the secret seed that
# generates your private key from your original account that is funding the
```

(continues on next page)

(continued from previous page)

```
# new account in the create account operation. You'll need the seed in order
# to sign off on the transaction. This is the source account.
old_account_seed = "SCVLSUGYEAUC4MVWJORB63JBMY2CEX6ATTJ5MXTENGD3IELUQF4F6HUB"
old_account_keypair = Keypair.from_seed(oldAccountSeed)

# This is the new account ID (the StrKey representation of your newly
# created public key). This is the destination account.
new_account_addr = "GXXX"

amount = '1' # Your new account minimum balance (in XLM) to transfer over
# create the CreateAccount operation
op = CreateAccount({
    'destination': new_account_addr,
    'starting_balance': amount
})
# create a memo
memo = TextMemo('Transferring to my new account!')

# Get the current sequence of the source account by contacting Horizon. You
# should also check the response for errors!
# Python 3
sequence = horizon.account(old_account_keypair.address().decode()).get('sequence')
# Python 2
# sequence = horizon.account(old_account_keypair.address()).get('sequence')

# Create a transaction with our single create account operation, with the
# default fee of 100 stroops as of this writing (0.00001 XLM)
tx = Transaction(
    source=kp.address().decode(),
    opts={
        'sequence': sequence,
        'memo': memo,
        'operations': [
            op,
        ],
    },
)
# Build a transaction envelope, ready to be signed.
envelope = Te(tx=tx, opts={"network_id": "PUBLIC"})

# Sign the transaction envelope with the source keypair
envelope.sign(old_account_keypair)

# Submit the transaction to Horizon
te_xdr = envelope.xdr()
response = horizon.submit(te_xdr)
```

Make sure to look at the response body carefully, as it can be an error or a successful response.

### 1.2.4 Looking up Account Details on Horizon

#### Basic Information

Once you have the account, you might want to look up its information from Horizon to verify the network knows about your new account:

```
from stellar_base.address import Address
publickey = 'GDVDKQFP665JAO7A2LSHNLQIUNYNAAIGJ6FYJVMG4DT3YJQQJSRBLQDG'
address = Address(address=publickey) # See signature for additional args
address.get() # Get the latest information from Horizon
```

You can now retrieve information for the account's

- Sequence Number
- Balances
- Paging Token
- Thresholds
- Flags
- Signers
- Data

Like so:

```
print("Balances: {}'.format(address.balances))
print("Sequence Number: {}'.format(address.sequence))
print("Flags: {}'.format(address.flags))
print("Signers: {}'.format(address.signers))
print("Data: {}'.format(address.data))
```

### Most Recent Payments

We can check the most recent payments by:

```
payments = address.payments()
```

Like many Horizon endpoints, payments is paginated. You can get different payments by using the following query parameters: limit, order, and cursor.

So if you need to check payments after a specific cursor, try:

```
address.payments(cursor='4225135422738433', limit=20, order='asc')
```

You can also use server sent events if you want to by passing in sse=True on methods that have sse in their signature.

```
address.payments(sse=True, limit=100)
```

### Other Account Attributes

Just like payments, there are plenty of other account attributes you can look up via Horizon:

```
address.transactions()
address.effects()
address.offers()
address.operations()
```

Look at the Horizon API reference for which endpoints support SSE.

### 1.2.5 Building A Transaction

When we created an account, we already created a transaction. We can build a transaction with a *Builder*, or with a *Transaction* object by itself. We recommend you use the builder, as it handles a lot of the details for you, and you can focus on the important parameters in each method's signature.

#### Using a Builder

Let's send Bob a payment that we owe him. We'd go about this in the following way:

```
from stellar_base.builder import Builder
seed = "SCVLSUGYEAUC4MVWJORB63JBMY2CEX6ATTJ5MXTENGD3IELUQF4F6HUB"
builder = Builder(secret=seed)
# builder = Builder(secret=seed, network='public') for LIVENET

bob_address = 'GXXX'
builder.append_payment_op(bob_address, '100', 'XLM')
builder.add_text_memo('For beers') # string length <= 28 bytes
builder.sign()

# Uses an internal horizon instance to submit over the network
builder.submit()
```

Or if you want to pay him with CNY:

```
# This is a stellar issuing account ID for an anchor that issues CNY
CNY_ISSUER = 'GDVDKQFP665JAO7A2LSHNLQIUNYNAAIGJ6FYJVMG4DT3YJQQJSRBLQDG'
builder.append_payment_op(bob_address, '10', 'CNY', CNY_ISSUER)
builder.add_text_memo('For beers') # string length <= 28 bytes
builder.sign()

# Uses an internal horizon instance to submit over the network
builder.submit()
```

And that's it!

Sometimes, we work with multi-signature transactions that require your signature in addition to the the account that originally sealed the transaction in an envelope. Typically you'll get an XDR string that you need to sign. To do this, you use *import_from_xdr* to import it into your builder.

```
# This is the transaction that you need to add your signature to
xdr_string = 'GXXX'
builder = Builder(secret=seed)
builder.import_from_xdr(xdr_string)
builder.sign()
xdr_string = builder.gen_xdr()
```

From here you can pass along your XDR string to anyone else who needs to sign it, or you can submit it via *builder.submit()* if you're the last to sign.

#### Using a Transaction Object

Here is a full example of how to make a Transaction from scratch. As you can see, it requires a lot more imports and knowledge of internal objects, but it gives you the most flexibility before submitting your transaction over the wire.

In this example, Alice is sending Bob 100 CNY.

```
from stellar_base.keypair import Keypair
from stellar_base.asset import Asset
from stellar_base.operation import Payment
from stellar_base.transaction import Transaction
from stellar_base.transaction_envelope import TransactionEnvelope as Te
from stellar_base.memo import TextMemo
from stellar_base.horizon import horizon_testnet, horizon_livenet

# Generate Alice's Keypair for ultimately signing and setting as the source
alice_seed = 'SAZJ3EDATROKTNNN4WZBZPRC34AN5WR43VEHAFKT5D66UEZTKDNKUHOK'
alice_kp = Keypair.from_seed(alice_seed)

# Bob's address, for the destination
bob_address = 'GDLP3SP4WP72L4BAJWZUDZ6SAYE4NAWILT5WQDS7RWC4XCUNUQDRB2A4'

# The CNY Issuer's address
CNY_ISSUER = 'GDVDKQFP665JAO7A2LSHNLQIUNYNAAIGJ6FYJVMG4DT3YJQQJSRBLQDG'

horizon = horizon_testnet()
# horizon = horizon_livenet() for LIVENET

# create op
amount = '100'
asset = Asset('CNY', CNY_ISSUER)
op = Payment({
    # Source is also inferred from the transaction source, so it's optional.
    'source' : alice_kp.address().decode(),
    'destination': bob_address,
    'asset': asset,
    'amount': amount
})
# create a memo
msg = TextMemo('For beers yesterday!')

# Get the current sequence of Alice
# Python 3
sequence = horizon.account(alice_kp.address().decode('utf-8')).get('sequence')
# Python 2
# sequence = horizon.account(alice_kp.address()).get('sequence')

# Construct a transaction
tx = Transaction(
    source = alice_kp.address().decode(),
    opts = {
        'sequence': sequence,
        'memo': msg,
        # Can specify a fee or use the default by not specifying it
        'fee': 100,
        'operations': [
            op,
        ],
    },
)


# Build transaction envelope
envelope = Te(tx=tx, opts={"network_id": "TESTNET"}) # or 'PUBLIC'
```

(continues on next page)

(continued from previous page)

```
# Sign the envelope
envelope.sign(alice_kp)

# Submit the transaction to Horizon!
xdr = envelope.xdr()
response = horizon.submit(xdr)
```

### 1.2.6 What's Next

From here, we recommend you explore our *API Documentation*. In there you'll find out more about the various objects that represent concepts in Stellar, as well as some of the additional helper classes and functions that exist.

Happy Coding!

CHAPTER 2

# API Documentation

Here you'll find detailed documentation on specific functions, classes, and methods.

## 2.1 API Documentation

### 2.1.1 Classes

#### Address

**class** stellar_base.address.**Address**(*address=None*, *secret=None*, *network='TESTNET'*, *horizon=None*)

The *Address* object, which represents an address (public key) on Stellar's network.

An *Address* is initialized via a public key string, or derived via a secret seed. The network on which the account exists is also specified, as it is used to verify and set attributes via connecting to Horizon. It mostly exists as a helper class for Horizon operations on a given account ID.

**Parameters**

- **address** (*str*) – The address string that represents this *Address*.
- **secret** (*str*) – The secret seed string that is used to derive the address for this *Address*.
- **network** (*str*) – The network to connect to for verifying and retrieving additional attributes from. Must be either 'PUBLIC' or 'TESTNET'.
- **horizon** (Horizon) – The Horizon instance to use for connecting to for additional information for the account to which this address corresponds to.

**effects**(*sse=False*, ***kwargs*)

Retrieve the effects JSON from this instance's Horizon server.

Retrieve the effects JSON response for the account associated with this *Address*.

**Parameters** **sse** (*bool*) – Use the SSE client for connecting to Horizon.

**get**()
Retrieve the account data that corresponds to this *Address*.

Retrieve the account data from Horizon for the account that corresponds to this *Address*. Attempt to retrieve the following attributes from Horizon:

- Sequence Number
- Balances
- Paging Token
- Thresholds
- Flags
- Signers
- Data

**Raises**

- **AccountNotExistError** – If the account does not exist, shown by a 404 response from a Horizon server.
- **Exception** – If any other problems come up, or if a network connection happens.

**offers**(***kwargs*)
Retrieve the offers JSON from this instance's Horizon server.

Retrieve the offers JSON response for the account associated with this *Address*.

**Parameters** **sse** (*bool*) – Use the SSE client for connecting to Horizon.

**operations**(*sse=False*, ***kwargs*)
Retrieve the operations JSON from this instance's Horizon server.

Retrieve the operations JSON response for the account associated with this *Address*.

**Parameters** **sse** (*bool*) – Use the SSE client for connecting to Horizon.

**payments**(*sse=False*, ***kwargs*)
Retrieve the payments JSON from this instance's Horizon server.

Retrieve the payments JSON response for the account associated with this *Address*.

**Parameters** **sse** (*bool*) – Use the SSE client for connecting to Horizon.

**transactions**(*sse=False*, ***kwargs*)
Retrieve the transactions JSON from this instance's Horizon server.

Retrieve the transactions JSON response for the account associated with this *Address*.

**Parameters** **sse** (*bool*) – Use the SSE client for connecting to Horizon.

## Asset

**class** stellar_base.asset.**Asset**(*code*, *issuer=None*)
The *Asset* object, which represents an asset and its corresponding issuer on the Stellar network.

For more information about the formats used for asset codes and how issuers work on Stellar's network, see Stellar's guide on assets.

**Parameters**

- **code** (*str*) – The asset code, in the formats specified in Stellar's guide on assets.
- **issuer** (*str*) – The strkey encoded issuer of the asset. Note if the currency is the native currency (XLM (Lumens)), no issuer is necessary.

**classmethod from_xdr**(*xdr*)
Create an *Asset* object from its base64 encoded XDR representation.

**Parameters** **xdr** (*bytes*) – The base64 encoded XDR Asset object.

**Returns** A new *Asset* object from its encoded XDR representation.

**classmethod from_xdr_object**(*asset_xdr_object*)
Create a *Asset* from an XDR Asset object.

**Parameters** **asset_xdr_object** – The XDR Asset object.

**Returns** A new *Asset* object from the given XDR Asset object.

**is_native**()
Return true if the *Asset* is the native asset.

**Returns** True if the Asset is native, False otherwise.

**static native**()
Create a *Asset* with the native currency.

Currently, the native currency is Stellar Lumens (XLM)

**Returns** A new *Asset* representing the native currency on the Stellar network.

**to_dict**()
Generate a dict for this object's attributes.

**Returns** A dict representing an *Asset*

**to_xdr_object**()
Create an XDR object for this *Asset*.

**Returns** An XDR Asset object

**xdr**()
Create an base64 encoded XDR string for this *Asset*.

**Return str** A base64 encoded XDR object representing this *Asset*.

### Builder

**class** stellar_base.builder.**Builder**(*secret=None*, *address=None*, *horizon=None*, *network=None*, *sequence=None*, *fee=100*)
The *Builder* object, which uses the builder pattern to create a list of operations in a `Transaction`, ultimately to be submitted as a `TransactionEnvelope` to the network via Horizon (see `Horizon`).

**Parameters**

- **secret** (*str*) – The base32 secret seed for the source address.
- **address** (*str*) – The base32 source address.
- **horizon** (`Horizon`) – The horizon instance to use for submitting the created transaction.
- **network** (*str*) – The network string that describes which version of Horizon to use, either the live net ('PUBLIC') or the test net ('TESTNET'). Defaults to TESTNET if an instance of Horizon has not been passed to the horizon param.

- **sequence** (*int*) – The sequence number to use for submitting this transaction with (must be the *current* sequence number of the source account)
- **fee** (*int*) – The network base fee is currently set to 100 stroops (0.00001 lumens). Transaction fee is equal to base fee times number of operations in this transaction.

**add_hash_memo**(*memo_hash*)
Set the memo for the transaction to a new *HashMemo*.

**Parameters** **memo_hash** (*bytes*) – A 32 byte hash to use as the memo.

**Returns** This builder instance.

**add_id_memo**(*memo_id*)
Set the memo for the transaction to a new *IdMemo*.

**Parameters** **memo_id** (*int*) – A 64 bit unsigned integer to set as the memo.

**Returns** This builder instance.

**add_memo**(*memo*)
Set the memo for the transaction build by this *Builder*.

**Parameters** **memo** (*Memo*) – A memo to add to this transaction.

**Returns** This builder instance.

**add_ret_hash_memo**(*memo_return*)
Set the memo for the transaction to a new *RetHashMemo*.

**Parameters** **memo_return** (*bytes*) – A 32 byte hash intended to be interpreted as the hash of the transaction the sender is refunding.

**Returns** This builder instance.

**add_text_memo**(*memo_text*)
Set the memo for the transaction to a new *TextMemo*.

**Parameters** **memo_text** (*str*) – The text for the memo to add.

**Returns** This builder instance.

**add_time_bounds**(*time_bounds*)
Add a time bound to this transaction.

Add a the UNIX timestamp, determined by ledger time, of a lower and upper bound of when this transaction will be valid. If a transaction is submitted too early or too late, it will fail to make it into the transaction set. maxTime equal 0 means that it's not set.

**Parameters** **time_bounds** (*list*) – A list of two Unix timestamps representing the lower and upper bound of when a given transaction will be valid.

**Returns** This builder instance.

**append_account_merge_op**(*destination*, *source=None*)
Append a *AccountMerge* operation to the list of operations.

**Parameters**

- **destination** (*str*) – The ID of the offer. 0 for new offer. Set to existing offer ID to update or delete.
- **source** (*str*) – The source address that is being merged into the destination account.

**Returns** This builder instance.

**append_allow_trust_op**(*trustor*, *asset_code*, *authorize*, *source=None*)
Append an *AllowTrust* operation to the list of operations.

**Parameters**

- **trustor** (*str*) – The account of the recipient of the trustline.
- **asset_code** (*str*) – The asset of the trustline the source account is authorizing. For example, if an anchor wants to allow another account to hold its USD credit, the type is USD:anchor.
- **authorize** (*bool*) – Flag indicating whether the trustline is authorized.
- **source** (*str*) – The source address that is establishing the trust in the allow trust operation.

**Returns** This builder instance.

**append_create_account_op**(*destination*, *starting_balance*, *source=None*)
Append a *CreateAccount* operation to the list of operations.

**Parameters**

- **destination** (*str*) – Account address that is created and funded.
- **starting_balance** (*int*) – Amount of XLM to send to the newly created account. This XLM comes from the source account.
- **source** (*str*) – The source address to deduct funds from to fund the new account.

**Returns** This builder instance.

**append_create_passive_offer_op**(*selling_code*, *selling_issuer*, *buying_code*, *buying_issuer*, *amount*, *price*, *source=None*)
Append a *CreatePassiveOffer* operation to the list of operations.

**Parameters**

- **selling_code** (*str*) – The asset code for the asset the offer creator is selling.
- **selling_issuer** (*str*) – The issuing address for the asset the offer creator is selling.
- **buying_code** (*str*) – The asset code for the asset the offer creator is buying.
- **buying_issuer** (*str*) – The issuing address for the asset the offer creator is selling.
- **amount** (*int*) – Amount of the asset being sold. Set to 0 if you want to delete an existing offer.
- **price** (*float*) – Decimal representation of the price of 1 unit of selling in terms of buying. For example, if you wanted to sell 30 XLM and buy 5 BTC, the price would be (5 / 30). Note that this does not take a tuple/dict with a numerator/denominator at this time.
- **source** (*str*) – The source address that is creating a passive offer on Stellar's distributed exchange.

**Returns** This builder instance.

**append_hashx_signer**(*hashx*, *signer_weight*, *source=None*)
Add a HashX signer to an account.

Add a HashX signer to an account via a `SetOptions <stellar_base.operation.SetOptions` operation. This is a helper function for *append_set_options_op()*.

**Parameters**

- **hashx** (*str*) – The address of the new hashX signer.

- **signer_weight** (*int*) – The weight of the new signer.
- **source** (*str*) – The source account that is adding a signer to its list of signers.

**Returns** This builder instance.

**append_inflation_op**(*source=None*)
Append a *Inflation* operation to the list of operations.

**Parameters** **source** (*str*) – The source address that is running the inflation operation.

**Returns** This builder instance.

**append_manage_data_op**(*data_name*, *data_value*, *source=None*)
Append a *ManageData* operation to the list of operations.

**Parameters**

- **data_name** (*str*) – String up to 64 bytes long. If this is a new Name it will add the given name/value pair to the account. If this Name is already present then the associated value will be modified.
- **data_value** (*str, bytes, None*) – If not present then the existing Name will be deleted. If present then this value will be set in the DataEntry. Up to 64 bytes long.
- **source** (*str*) – The source account on which data is being managed. operation.

**Returns** This builder instance.

**append_manage_offer_op**(*selling_code*, *selling_issuer*, *buying_code*, *buying_issuer*, *amount*, *price*, *offer_id=0*, *source=None*)
Append a *ManageOffer* operation to the list of operations.

**Parameters**

- **selling_code** (*str*) – The asset code for the asset the offer creator is selling.
- **selling_issuer** (*str*) – The issuing address for the asset the offer creator is selling.
- **buying_code** (*str*) – The asset code for the asset the offer creator is buying.
- **buying_issuer** (*str*) – The issuing address for the asset the offer creator is selling.
- **amount** (*int*) – Amount of the asset being sold. Set to 0 if you want to delete an existing offer.
- **price** (*float*) – Decimal representation of the price of 1 unit of selling in terms of buying. For example, if you wanted to sell 30 XLM and buy 5 BTC, the price would be (5 / 30). Note that this does not take a tuple/dict with a numerator/denominator at this time.
- **offer_id** (*str*) – The ID of the offer. 0 for new offer. Set to existing offer ID to update or delete.
- **source** (*str*) – The source address that is managing an offer on Stellar's distributed exchange.

**Returns** This builder instance.

**append_op**(*operation*)
Append an *Operation* to the list of operations.

Add the operation specified if it doesn't already exist in the list of operations of this *Builder* instance.

**Parameters** **operation** (Operation) – The operation to append to the list of operations.

**Returns** This builder instance.

**append_path_payment_op**(*destination*, *send_code*, *send_issuer*, *send_max*, *dest_code*, *dest_issuer*, *dest_amount*, *path*, *source=None*)

Append a `PathPayment` operation to the list of operations.

**Parameters**

- **destination** (*str*) – The destination address (Account ID) for the payment.
- **send_code** (*str*) – The asset code for the source asset deducted from the source account.
- **send_issuer** (*str*) – The address of the issuer of the source asset.
- **send_max** (*int*) – The maximum amount of send asset to deduct (excluding fees).
- **dest_code** (*str*) – The asset code for the final destination asset sent to the recipient.
- **dest_issuer** (*str*) – Account address that receives the payment.
- **dest_amount** (*str*) – The amount of destination asset the destination account receives.
- **path** (*list*) – A list of asset tuples, each tuple containing a (code, issuer_address) for each asset in the path. For the native asset, an empty string is used for the issuer address.
- **source** (*str*) – The source address of the path payment.

**Returns** This builder instance.

**append_payment_op**(*destination*, *amount*, *asset_code='XLM'*, *asset_issuer=None*, *source=None*)

Append a `Payment` operation to the list of operations.

**Parameters**

- **destination** (*str*) – Account address that receives the payment.
- **amount** (*int*) – The amount of the currency to send in the payment.
- **asset_code** (*str*) – The asset code for the asset to send.
- **asset_issuer** (*str*) – The address of the issuer of the asset.
- **source** (*str*) – The source address of the payment.

**Returns** This builder instance.

**append_pre_auth_tx_signer**(*pre_auth_tx*, *signer_weight*, *source=None*)

Add a PreAuthTx signer to an account.

Add a PreAuthTx signer to an account via a `SetOptions <stellar_base.operation.SetOptions` operation. This is a helper function for `append_set_options_op()`.

**Parameters**

- **pre_auth_tx** (*str*) – The address of the new preAuthTx signer - obtained by calling *hash_meta* on the TransactionEnvelope.
- **signer_weight** (*int*) – The weight of the new signer.
- **source** (*str*) – The source account that is adding a signer to its list of signers.

**Returns** This builder instance.

**append_set_options_op**(*inflation_dest=None*, *clear_flags=None*, *set_flags=None*, *master_weight=None*, *low_threshold=None*, *med_threshold=None*, *high_threshold=None*, *home_domain=None*, *signer_address=None*, *signer_type=None*, *signer_weight=None*, *source=None*)

Append a `SetOptions` operation to the list of operations.

**Parameters**

- **inflation_dest** (*str*) – The address in which to send inflation to on an *Inflation* operation.
- **clear_flags** (*int*) – Indicates which flags to clear. For details about the flags, please refer to Stellar's documentation on Accounts. The bit mask integer subtracts from the existing flags of the account. This allows for setting specific bits without knowledge of existing flags.
- **set_flags** (*int*) – Indicates which flags to set. For details about the flags, please refer to Stellar's documentation on Accounts. The bit mask integer adds onto the existing flags of the account. This allows for setting specific bits without knowledge of existing flags.
- **master_weight** (*int*) – Weight of the master key. This account may also add other keys with which to sign transactions using the signer param.
- **low_threshold** (*int*) – A number from 0-255 representing the threshold this account sets on all operations it performs that have a low threshold.
- **med_threshold** (*int*) – A number from 0-255 representing the threshold this account sets on all operations it performs that have a medium threshold.
- **high_threshold** (*int*) – A number from 0-255 representing the threshold this account sets on all operations it performs that have a high threshold.
- **home_domain** (*str*) – Sets the home domain of an account. See Stellar's documentation on Federation.
- **signer_address** (*str*) – The address of the new signer to add to the source account.
- **signer_type** (*str*) – The type of signer to add to the account. Must be in ('ed25519PublicKey', 'hashX', 'preAuthTx'). See Stellar's documentation for Multi-Sign for more information.
- **signer_weight** (*int*) – The weight of the signer. If the weight is 0, the signer will be deleted.
- **source** (*str*) – The source address for which options are being set.

**Returns** This builder instance.

**append_trust_op**(*destination*, *code*, *limit=None*, *source=None*)

Append a *ChangeTrust* operation to the list of operations.

**Parameters**

- **destination** (*str*) – The issuer address for the asset.
- **code** (*str*) – The asset of the trustline. For example, if a user extends a trustline of up to 200 USD to an anchor, the line is USD:anchor.
- **limit** (*str*) – The limit of the new trustline.
- **source** (*str*) – The source address to add the trustline to.

**Returns** This builder instance.

**federation_payment**(*fed_address*, *amount*, *asset_code='XLM'*, *asset_issuer=None*, *source=None*)

Append a *Payment* operation to the list of operations using federation on the destination address.

Translates the destination stellar address to an account ID via *federation*, before creating a new payment operation via *append_payment_op()*.

**Parameters**

- **fed_address** (*str*) – A Stellar Address that needs to be translated into a valid account ID via federation.
- **amount** (*int*) – The amount of the currency to send in the payment.
- **asset_code** (*str*) – The asset code for the asset to send.
- **asset_issuer** (*str*) – The address of the issuer of the asset.
- **source** (*str*) – The source address of the payment.

**Returns** This builder instance.

**gen_compliance_xdr**()
Create an XDR object representing this builder's transaction to be sent over via the Compliance protocol (notably, with a sequence number of 0).

Intentionally, the XDR object is returned without any signatures on the transaction.

See Stellar's documentation on its Compliance Protocol for more information.

**gen_te**()
Generate a *TransactionEnvelope* around the generated Transaction via the list of operations in this instance.

**Returns** A transaction envelope ready to send over the network.

**Return type** *TransactionEnvelope*

**gen_tx**()
Generate a *Transaction* object from the list of operations contained within this object.

**Returns** A transaction representing all of the operations that have been appended to this builder.

**Return type** *Transaction*

**gen_xdr**()
Create an XDR object around a newly generated *TransactionEnvelope*.

**Returns** An XDR object representing a newly created transaction envelope ready to send over the network.

**get_sequence**()
Get the sequence number for a given account via Horizon.

**Returns** The current sequence number for a given account

**Return type** int

**import_from_xdr**(*xdr*)
Create a *TransactionEnvelope* via an XDR object.

In addition, sets the fields of this builder (the transaction envelope, transaction, operations, source, etc.) to all of the fields in the provided XDR transaction envelope.

**Parameters** **xdr** – The XDR object representing the transaction envelope to which this builder is setting its state to.

**next_builder**()
Create a new builder based off of this one with its sequence number incremented.

**Returns** A new Builder instance

**Return type** *Builder*

**sign**(*secret=None*)
Sign the generated *TransactionEnvelope* from the list of this builder's operations.

**Parameters** **secret** (*str*) – The secret seed to use if a key pair or secret was not provided when this class was originaly instantiated, or if another key is being utilized to sign the transaction envelope.

**sign_preimage**(*preimage*)
Sign the generated transaction envelope using a Hash(x) signature.

**Parameters** **preimage** (*str*) – The value to be hashed and used as a signer on the transaction envelope.

**submit**()
Submit the generated XDR object of the built transaction envelope to Horizon.

Sends the generated transaction envelope over the wire via this builder's *Horizon* instance. Note that you'll typically want to sign the transaction before submitting via the sign methods.

**Returns** A dict representing the JSON response from Horizon.

**Raises** HTTPError

### Keypair

**class** stellar_base.keypair.**Keypair**(*verifying_key*, *signing_key=None*)
The *Keypair* object, which represents a signing and verifying key for use with the Stellar network.

Instead of instantiating the class directly, we recommend using one of several class methods:

- *Keypair.random()*
- *Keypair.deterministic()*
- *Keypair.from_seed()*
- *Keypair.from_address()*

**Parameters**

- **verifying_key** (*ed25519.VerifyingKey*) – The verifying (public) Ed25519 key in the keypair.
- **signing_key** (*ed25519.SigningKey*) – The signing (private) Ed25519 key in the keypair.

**account_xdr_object**()
Create PublicKey XDR object via public key bytes.

**Returns** Serialized XDR of PublicKey type.

**address**()
Get the public key encoded as a strkey.

See encode_check() for more details on the strkey encoding process.

**Returns** The public key encoded as a strkey.

**Return type** str

**classmethod deterministic**(*mnemonic*, *passphrase=''*, *lang='english'*, *index=0*)
Generate a *Keypair* object via a deterministic phrase.

Using a mnemonic, such as one generated from StellarMnemonic, generate a new keypair deterministically. Uses StellarMnemonic internally to generate the seed from the mnemonic, using PBKDF2.

**Parameters**

- **mnemonic** (*str*) – A unique string used to deterministically generate keypairs.
- **passphrase** (*str*) – An optional passphrase used as part of the salt during PBKDF2 rounds when generating the seed from the mnemonic.
- **lang** (*str*) – The language of the mnemonic, defaults to english.
- **index** (*int*) – The index of the keypair generated by the mnemonic. This allows for multiple Keypairs to be derived from the same mnemonic, such as:

```
>>> from stellar_base import Keypair
>>> m = 'hello world'  # Don't use this mnemonic in practice.
>>> kp1 = Keypair.deterministic(m, lang='english', index=0)
>>> kp2 = Keypair.deterministic(m, lang='english', index=1)
>>> kp3 = Keypair.deterministic(m, lang='english', index=2)
```

**Returns** A new *Keypair* instance derived from the mnemonic.

**classmethod from_address**(*address*)
Generate a *Keypair* object via a strkey encoded public key.

**Parameters** **address** (*str*) – A base32 encoded public key encoded as described in `encode_check()`

**Returns** A new *Keypair* with only a verifying (public) key.

**classmethod from_base58_seed**(*base58_seed*)
Generate a *Keypair* object via Base58 encoded seed.

Deprecated since version 0.1.7: Base58 address encoding is DEPRECATED! Use this method only for transition to strkey encoding.

**Parameters** **base58_seed** (*str*) – A base58 encoded encoded secret seed.

**Returns** A new *Keypair* derived from the secret seed.

**classmethod from_raw_seed**(*raw_seed*)
Generate a *Keypair* object via a sequence of bytes.

Typically these bytes are random, such as the usage of `os.urandom()` in *Keypair.random()*. However this class method allows you to use an arbitrary sequence of bytes, provided the sequence is 32 bytes long.

**Parameters** **raw_seed** (*bytes*) – A bytes object used as the seed for generating the keypair.

**Returns** A new *Keypair* derived by the raw secret seed.

**classmethod from_seed**(*seed*)
Generate a *Keypair* object via a strkey encoded seed.

**Parameters** **seed** (*str*) – A base32 encoded secret seed string encoded as described in `encode_check()`.

**Returns** A new *Keypair* instance derived by the secret seed.

**public_key**()
See *Keypair.account_xdr_object()*.

**classmethod random**()
Generate a *Keypair* object via a randomly generated seed.

**raw_public_key**()
Get the bytes that comprise the verifying (public) key.

**Returns** The verifying key's bytes.

**raw_seed**()
Get the bytes of the signing key's seed.

**Returns** The signing key's secret seed as a byte sequence.

**Return type** bytes

**seed**()
Get the secret seed encoded as a strkey.

See `encode_check()` for more details on the strkey encoding process.

**Returns** The secret seed encoded as a strkey.

**Return type** str

**sign**(*data*)
Sign a bytes-like object using the signing (private) key.

**Parameters** **data** (*bytes*) – The data to sign

**Returns** The signed data

**Return type** bytes

**sign_decorated**(*data*)
Sign a bytes-like object and return the decorated signature.

Sign a bytes-like object by signing the data using the signing (private) key, and return a decorated signature, which includes the last four bytes of the public key as a signature hint to go along with the signature as an XDR DecoratedSignature object.

**Parameters** **data** (*bytes*) – A sequence of bytes to sign, typically a transaction.

**signature_hint**()
Get the signature hint derived from the public key in this *Keypair*.

Get the last four bytes of the public key to be used as a signature hint when utilizing decorated signatures.

**verify**(*data*, *signature*)
Verify the signature of a sequence of bytes.

Verify the signature of a sequence of bytes using the verifying (public) key and the data that was originally signed, otherwise throws an exception.

**Parameters**

- **data** (*bytes*) – A sequence of bytes that were previously signed by the private key associated with this verifying key.
- **signature** (*bytes*) – A sequence of bytes that comprised the signature for the corresponding data.

**xdr**()
Generate base64 encoded XDR PublicKey object.

Return a base64 encoded PublicKey XDR object, for sending over the wire when interacting with stellard.

**Returns** The base64 encoded PublicKey XDR structure.

### Memo

**class** stellar_base.memo.**Memo**
The *Memo* object, which represents the base class for memos for use with Stellar transactions.

The memo for a transaction contains optional extra information about the transaction taking place. It is the responsibility of the client to interpret this value.

See the following implementations that serve a more practical use with the library:

- *NoneMemo* - No memo.
- ***TextMemo*** **- A string encoded using either ASCII or UTF-8, up to** 28-bytes long.
- *IdMemo* - A 64 bit unsigned integer.
- *HashMemo* - A 32 byte hash.
- ***RetHashMemo*** **- A 32 byte hash intended to be interpreted as the** hash of the transaction the sender is refunding.

See Stellar's documentation on Transactions for more information on how memos are used within transactions, as well as information on the available types of memos.

**to_xdr_object**()
Creates an XDR Memo object that represents this *Memo*.

**xdr**()
Packs and base64 encodes this *Memo* as an XDR string.

**class** stellar_base.memo.**NoneMemo**
The *NoneMemo*, which represents no memo for a transaction.

**to_xdr_object**()
Creates an XDR Memo object for a transaction with no memo.

**class** stellar_base.memo.**TextMemo**(*text*)
The *TextMemo*, which represents MEMO_TEXT in a transaction.

**Parameters** **text** (*str*) – A string encoded using either ASCII or UTF-8, up to 28-bytes long.

**to_xdr_object**()
Creates an XDR Memo object for a transaction with MEMO_TEXT.

**class** stellar_base.memo.**IdMemo**(*memo_id*)
The *IdMemo* which represents MEMO_ID in a transaction.

**Parameters** **memo_id** (*int*) – A 64 bit unsigned integer.

**to_xdr_object**()
Creates an XDR Memo object for a transaction with MEMO_ID.

**class** stellar_base.memo.**HashMemo**(*memo_hash*)
The *HashMemo* which represents MEMO_HASH in a transaction.

**Parameters** **memo_hash** (*bytes*) – A 32 byte hash.

**to_xdr_object**()
Creates an XDR Memo object for a transaction with MEMO_HASH.

**class** stellar_base.memo.**RetHashMemo**(*memo_return*)
The *RetHashMemo* which represents MEMO_RETURN in a transaction.

MEMO_RETURN is typically used with refunds/returns over the network - it is a 32 byte hash intended to be interpreted as the hash of the transaction the sender is refunding.

**Parameters** **memo_return** (*bytes*) – A 32 byte hash intended to be interpreted as the hash of the transaction the sender is refunding.

**to_xdr_object**()
Creates an XDR Memo object for a transaction with MEMO_RETURN.

### Network

**class** stellar_base.network.**Network**(*passphrase=None*)
The *Network* object, which represents a Stellar network.

This class represents such a stellar network such as the public livenet and the Stellar Development Foundation Test network.

**Parameters** **passphrase** (*str*) – The passphrase for the network

**network_id**()
Get the network ID of the network.

Get the network ID of the network, which is an XDR hash of the passphrase.

stellar_base.network.**live_network**()
Get the *Network* representing the live Network.

stellar_base.network.**test_network**()
Get the *Network* representing the Test Network.

### Operation

**class** stellar_base.operation.**Operation**(*opts*)
The *Operation* object, which represents an operation on Stellar's network.

An operation is an individual command that mutates Stellar's ledger. It is typically rolled up into a transaction (a transaction is a list of operations with additional metadata).

Operations are executed on behalf of the source account specified in the transaction, unless there is an override defined for the operation.

For more on operations, see Stellar's documentation on operations as well as Stellar's List of Operations, which includes information such as the security necessary for a given operation, as well as information about when validity checks occur on the network.

The *Operation* class is typically not used, but rather one of its subclasses is typically included in transactions.

**Parameters** **opts** (*dict*) – A dict of options for creating this *Operation*. By default, this only pulls out the source account via opts.source.

**classmethod** **from_xdr**(*xdr*)
Create the appropriate *Operation* subclass from the XDR structure.

Decode an XDR base64 encoded string and create the appropriate *Operation* object.

**Parameters** **xdr** (*str*) – The XDR object to create an *Operation* (or subclass) instance from.

**static** **from_xdr_amount**()
Converts an amount from an XDR object into its appropriate integer representation.

Each asset amount is encoded as a signed 64-bit integer in the XDR structures. An asset amount unit (that which is seen by end users) is scaled down by a factor of ten million (10,000,000) to arrive at the native 64-bit integer representation. For example, the integer amount value 25,123,456 equals 2.5123456 units of the asset. This scaling allows for seven decimal places of precision in human-friendly amount units.

This static method correctly divides the value by the scaling factor in order to get the proper units of the asset.

See Stellar's documentation on Asset Precision for more information.

**Parameters** **value** (*int*) – The amount to convert to a string from an XDR int64 amount.

**static to_xdr_amount**()
Converts an amount to the appropriate value to send over the network as a part of an XDR object.

Each asset amount is encoded as a signed 64-bit integer in the XDR structures. An asset amount unit (that which is seen by end users) is scaled down by a factor of ten million (10,000,000) to arrive at the native 64-bit integer representation. For example, the integer amount value 25,123,456 equals 2.5123456 units of the asset. This scaling allows for seven decimal places of precision in human-friendly amount units.

This static method correctly multiplies the value by the scaling factor in order to come to the integer value used in XDR structures.

See Stellar's documentation on Asset Precision for more information.

**Parameters** **value** (*str*) – The amount to convert to an integer for XDR serialization.

**to_xdr_object**()
Creates an XDR Operation object that represents this *Operation*.

**xdr**()
Packs and base64 encodes this *Operation* as an XDR string.

### Transaction

**class** stellar_base.transaction.**Transaction**(*source*, *opts*)
The *Transaction* object, which represents a transaction on Stellar's network.

A transaction contains a list of operations (see *Operation*), which are all executed in order as one ACID transaction, along with an associated source account, fee, account sequence number, list of signatures, both an optional memo and an optional timebound. Typically a *Transaction* is placed in a *TransactionEnvelope* which is then signed before being sent over the network.

For more information on Transactions in Stellar, see Stellar's guide on transactions.

**Parameters**

- **source** (*str*) – A strkey encoded account ID (public key) of the source account for the transaction.
- **opts** (*dict*) – A dict that contains the primary components of the transaction. The following keys are searched from the dict:
  - opts.sequence - The current sequence number of the source account. The sequence number is incremented by 1 automatically, as is necessary for a new transaction for a given source account.
  - opts.time_bounds - A list of two Unix timestamps representing the lower and upper bound of when a given transaction will be valid. NOTE: This should likely be an object containing a minTime and maxTime attribute instead, and the implementation may change.
  - opts.memo - The memo being sent with the transaction, being represented as one of the subclasses of the *Memo* object. Defaults to *NoneMemo*.
  - opts.fee - The fee amount for the transaction, which should equal BASE_FEE (currently 100 stroops) multiplied by the number of operations in the transaction. See Stellar's latest documentation on fees for more information.
  - opts.operations: A list of *Operation* objects (typically its subclasses as defined in stellar_base.operation to be included in the transaction. By default this is an empty list.

**add_operation**(*operation*)
Add an *Operation* to this transaction.

This method will only add an operation if it is not already in the transaction's list of operations, i.e. every operation in the transaction should be unique.

**Parameters** **operation** (Operation) – The operation to add to this transaction.

**classmethod from_xdr_object**(*tx_xdr_object*)
Create a *Transaction* object from a Transaction XDR object.

**to_xdr_object**()
Creates an XDR Transaction object that represents this *Transaction*.

**xdr**()
Packs and base64 encodes this *Transaction* as an XDR string.

### TransactionEnvelope

**class** stellar_base.transaction_envelope.**TransactionEnvelope**(*tx*, *opts=None*)
The *TransactionEnvelope* object, which represents a transaction envelope ready to sign and submit to send over the network.

When a transaction is ready to be prepared for sending over the network, it must be put into a *TransactionEnvelope*, which includes additional metadata such as the signers for a given transaction. Ultimately, this class handles signing and conversion to and from XDR for usage on Stellar's network.

**Parameters**

- **tx** (*Transaction*) – The transaction that is encapsulated in this envelope.
- **opts** (*dict*) – Additional options, such as:
  - opts.signatures, which contains a list of signatures that have already been created.
  - opts.network_id, which contains the network ID for which network this transaction envelope is associated with.

**classmethod from_xdr**(*xdr*)
Create a new *TransactionEnvelope* from an XDR string.

**Parameters** **xdr** (*bytes*) – The XDR string that represents a transaction envelope.

**hash_meta**()
Get the XDR Hash of the signature base.

This hash is ultimately what is signed before transactions are sent over the network. See *signature_base()* for more details about this process.

**Returns** The XDR Hash of this transaction envelope's signature base.

**sign**(*keypair*)
Sign this transaction envelope with a given keypair.

Note that the signature must not already be in this instance's list of signatures.

**Parameters** **keypair** (*Keypair*) – The keypair to use for signing this transaction envelope.

**Raises** SignatureExistError

**sign_hashX**(*preimage*)
Sign this transaction envelope with a Hash(X) signature.

See Stellar's documentation on Multi-Sig for more details on Hash(x) signatures.

**Parameters** **preimage** (*str*) – The "x" value to be hashed and used as a signature.

**signature_base**()
Get the signature base of this transaction envelope.

Return the "signature base" of this transaction, which is the value that, when hashed, should be signed to create a signature that validators on the Stellar Network will accept.

It is composed of a 4 prefix bytes followed by the xdr-encoded form of this transaction.

**Returns** The signature base of this transaction envelope.

**to_xdr_object**()
Get an XDR object representation of this *TransactionEnvelope*.

**xdr**()
Get the base64 encoded XDR string representing this *TransactionEnvelope*.

### List of Operations

### Create Account

**class** stellar_base.operation.**CreateAccount**(*opts*)
The *CreateAccount* object, which represents a Create Account operation on Stellar's network.

This operation creates and funds a new account with the specified starting balance.

Threshold: Medium

**Parameters** **opts** (*dict*) – A dict of options for creating this *CreateAccount*. This class pulls a 'source', 'destination', and 'starting_balance' via opts.

**classmethod from_xdr_object**(*op_xdr_object*)
Creates a *CreateAccount* object from an XDR Operation object.

**to_xdr_object**()
Creates an XDR Operation object that represents this *CreateAccount*.

### Payment

**class** stellar_base.operation.**Payment**(*opts*)
The *Payment* object, which represents a Payment operation on Stellar's network.

Sends an amount in a specific asset to a destination account.

Threshold: Medium

**Parameters** **opts** (*dict*) – A dict of options for creating this *Payment*. This class pulls a 'source', 'destination', 'asset', and 'amount' via opts.

**classmethod from_xdr_object**(*op_xdr_object*)
Creates a *Payment* object from an XDR Operation object.

**to_xdr_object**()
Creates an XDR Operation object that represents this *Payment*.

### Path Payment

**class** stellar_base.operation.**PathPayment**(*opts*)

The *PathPayment* object, which represents a PathPayment operation on Stellar's network.

Sends an amount in a specific asset to a destination account through a path of offers. This allows the asset sent (e.g., 450 XLM) to be different from the asset received (e.g, 6 BTC).

Threshold: Medium

**Parameters** **opts** (*dict*) – A dict of options for creating this *PathPayment*. This class pulls a 'source', 'destination', 'send_asset', 'send_max', 'dest_asset', 'dest_amount', and 'path' via opts.

**classmethod from_xdr_object**(*op_xdr_object*)

Creates a *PathPayment* object from an XDR Operation object.

**to_xdr_object**()

Creates an XDR Operation object that represents this *PathPayment*.

### Manage Offer

**class** stellar_base.operation.**ManageOffer**(*opts*)

The *ManageOffer* object, which represents a ManageOffer operation on Stellar's network.

Creates, updates, or deletes an offer.

If you want to create a new offer set Offer ID to 0.

If you want to update an existing offer set Offer ID to existing offer ID.

If you want to delete an existing offer set Offer ID to existing offer ID and set Amount to 0.

Threshold: Medium

**Parameters** **opts** (*dict*) – A dict of options for creating this *ManageOffer*. This class pulls several of the following from opts: 'source', 'selling', 'buying', 'amount', 'price', 'offer_id'.

**classmethod from_xdr_object**(*op_xdr_object*)

Creates a *ManageOffer* object from an XDR Operation object.

**to_xdr_object**()

Creates an XDR Operation object that represents this *ManageOffer*.

### Create Passive Offer

**class** stellar_base.operation.**CreatePassiveOffer**(*opts*)

The *CreatePassiveOffer* object, which represents a CreatePassiveOffer operation on Stellar's network.

A passive offer is an offer that does not act on and take a reverse offer of equal price. Instead, they only take offers of lesser price. For example, if an offer exists to buy 5 BTC for 30 XLM, and you make a passive offer to buy 30 XLM for 5 BTC, your passive offer does not take the first offer.

Note that regular offers made later than your passive offer can act on and take your passive offer, even if the regular offer is of the same price as your passive offer.

Passive offers allow market makers to have zero spread. If you want to trade EUR for USD at 1:1 price and USD for EUR also at 1:1, you can create two passive offers so the two offers don't immediately act on each other.

Once the passive offer is created, you can manage it like any other offer using the manage offer operation - see *ManageOffer* for more details.

**Parameters** **opts** (*dict*) – A dict of options for creating this *CreatePassiveOffer*. This class pulls several of the following from opts: 'source', 'selling', 'buying', 'amount', 'price'.

**classmethod from_xdr_object**(*op_xdr_object*)
Creates a *CreatePassiveOffer* object from an XDR Operation object.

**to_xdr_object**()
Creates an XDR Operation object that represents this *CreatePassiveOffer*.

### Set Options

**class** stellar_base.operation.**SetOptions**(*opts*)
The *SetOptions* object, which represents a SetOptions operation on Stellar's network.

This operation sets the options for an account.

For more information on the signing options, please refer to the multi-sig doc.

When updating signers or other thresholds, the threshold of this operation is high.

Threshold: Medium or High

**Parameters** **opts** (*dict*) – A dict of options for creating this *SetOptions*. This class pulls several of the following depending on the option: a 'source', 'inflation_dest', 'clear_flags', 'set_flags', 'master_weight', 'low_threshold', 'med_threshold', 'high_threshold', 'home_domain', 'signer_address', 'signer_type', 'signer_weight' via opts.

**classmethod from_xdr_object**(*op_xdr_object*)
Creates a *SetOptions* object from an XDR Operation object.

**to_xdr_object**()
Creates an XDR Operation object that represents this *SetOptions*.

### Change Trust

**class** stellar_base.operation.**ChangeTrust**(*opts*)
The *ChangeTrust* object, which represents a ChangeTrust operation on Stellar's network.

Creates, updates, or deletes a trustline. For more on trustlines, please refer to the *assets documentation <https://www.stellar.org/developers/guides/concepts/assets.html>_*.

Threshold: Medium

**Parameters** **opts** (*dict*) – A dict of options for creating this *ChangeTrust*. This class pulls a 'source', 'asset', and optionally a 'limit' via opts.

**classmethod from_xdr_object**(*op_xdr_object*)
Creates a *ChangeTrust* object from an XDR Operation object.

**to_xdr_object**()
Creates an XDR Operation object that represents this *ChangeTrust*.

### Allow Trust

**class** stellar_base.operation.**AllowTrust**(*opts*)

The *AllowTrust* object, which represents a AllowTrust operation on Stellar's network.

Updates the authorized flag of an existing trustline. This can only be called by the issuer of a trustline's asset.

The issuer can only clear the authorized flag if the issuer has the AUTH_REVOCABLE_FLAG set. Otherwise, the issuer can only set the authorized flag.

Threshold: Low

**Parameters** **opts** (*dict*) – A dict of options for creating this *AllowTrust*. This class pulls a 'source', 'trustor', 'asset_code', and 'authorize' via opts.

**classmethod from_xdr_object**(*op_xdr_object*)

Creates a *AllowTrust* object from an XDR Operation object.

**to_xdr_object**()

Creates an XDR Operation object that represents this *AllowTrust*.

### Account Merge

**class** stellar_base.operation.**AccountMerge**(*opts*)

The *AccountMerge* object, which represents a AccountMerge operation on Stellar's network.

Transfers the native balance (the amount of XLM an account holds) to another account and removes the source account from the ledger.

Threshold: High

**Parameters** **opts** (*dict*) – A dict of options for creating this *AccountMerge*. This class pulls several of the following from opts: 'source', 'destination'

**classmethod from_xdr_object**(*op_xdr_object*)

Creates a *AccountMerge* object from an XDR Operation object.

**to_xdr_object**()

Creates an XDR Operation object that represents this *AccountMerge*.

### Inflation

**class** stellar_base.operation.**Inflation**(*opts*)

The *Inflation* object, which represents a Inflation operation on Stellar's network.

This operation runs inflation.

Threshold: Low

**Parameters** **opts** (*dict*) – A dict of options for creating this *Inflation*. This class pulls several of the following from opts: 'source'.

**classmethod from_xdr_object**(*op_xdr_object*)

Creates a *Inflation* object from an XDR Operation object.

**to_xdr_object**()

Creates an XDR Operation object that represents this *Inflation*.

#### Manage Data

**class** stellar_base.operation.**ManageData**(*opts*)
The *ManageData* object, which represents a ManageData operation on Stellar's network.

Allows you to set,modify or delete a Data Entry (name/value pair) that is attached to a particular account. An account can have an arbitrary amount of DataEntries attached to it. Each DataEntry increases the minimum balance needed to be held by the account.

DataEntries can be used for application specific things. They are not used by the core Stellar protocol.

Threshold: Medium

**Parameters** **opts** (*dict*) – A dict of options for creating this *ManageData*. This class pulls several of the following from opts: 'source', 'data_name', 'data_value'.

**classmethod from_xdr_object**(*op_xdr_object*)
Creates a *ManageData* object from an XDR Operation object.

**to_xdr_object**()
Creates an XDR Operation object that represents this *ManageData*.

### 2.1.2 Federation

**exception** stellar_base.federation.**FederationError**
A *FederationError* that represents an issue stemming from Stellar Federation.

stellar_base.federation.**federation**(*address_or_id*, *fed_type='name'*, *domain=None*, *allow_http=False*)
Send a federation query to a Stellar Federation service.

For more info, see the complete guide on Stellar Federation..

**Parameters**

- **address_or_id** (*str*) – The address which you expect te retrieve federation information about.
- **fed_type** (*str*) – The type of federation query that you are making. Must be 'name', 'id', 'forward', or 'txid'.
- **domain** (*str*) – The domain that corresponds to the address or ID; this is where the stellar.toml file lives (which ultimately points to the URL of the federation service).
- **allow_http** (*bool*) – Whether to allow for requests over plain HTTP over HTTPS. Note - you should *always* use HTTPS outside of testing.

**Return dict** The federation query response decoded from JSON as a dict.

stellar_base.federation.**get_auth_server**(*domain*, *allow_http=False*)
Retrieve the AUTH_SERVER config from a domain's stellar.toml.

**Parameters**

- **domain** (*str*) – The domain the .toml file is hosted at.
- **allow_http** (*bool*) – Specifies whether the request should go over plain HTTP vs HTTPS. Note it is recommend that you *always* use HTTPS.

**Return str** The AUTH_SERVER url.

stellar_base.federation.**get_federation_service**(*domain*, *allow_http=False*)
Retrieve the FEDERATION_SERVER config from a domain's stellar.toml.

**Parameters**

- **domain** (*str*) – The domain the .toml file is hosted at.
- **allow_http** (*bool*) – Specifies whether the request should go over plain HTTP vs HTTPS. Note it is recommend that you *always* use HTTPS.

**Return str** The FEDERATION_SERVER url.

stellar_base.federation.**get_stellar_toml**(*domain*, *allow_http=False*)

Retrieve the stellar.toml file from a given domain.

Retrieve the stellar.toml file for information about interacting with Stellar's federation protocol for a given Stellar Anchor (specified by a domain).

**Parameters**

- **domain** (*str*) – The domain the .toml file is hosted at.
- **allow_http** (*bool*) – Specifies whether the request should go over plain HTTP vs HTTPS. Note it is recommend that you *always* use HTTPS. :return: The stellar.toml file as a an object via `toml.loads()`.

### 2.1.3 Horizon

**class** stellar_base.horizon.**Horizon**(*horizon=None*, *sse=False*, *timeout=20*)

**account**(*address*, ***kwargs*)

Returns information and links relating to a single account.

GET /accounts/{account}

**Parameters** **address** (*str*) – The account ID to retrieve details about

**Returns** The account details in a JSON response

**Return type** dict

**account_data**(*account_id*, *data_key*, ***kwargs*)

This endpoint represents a single data associated with a given account.

GET /accounts/{account}/data/{key}

**Parameters**

- **account_id** (*str*) – The account ID to look up a data item from
- **data_key** (*str*) – The name of the key for the data item in question

**Returns** The value of the data field for the given account and data key

**Return type** dict

**account_effects**(*address*, *params=None*, *sse=None*, ***kwargs*)

This endpoint represents all effects that changed a given account.

GET /accounts/{account}/effects{?cursor,limit,order}

**Parameters**

- **address** (*str*) – The account ID to look up effects for.
- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.

- **sse** (*bool*) – Use server side events for streaming responses

**Returns** The list of effects in a JSON response.

**Return type** dict

**account_offers**(*address*, *params=None*, ***kwargs*)
This endpoint represents all the offers a particular account makes.

GET /accounts/{account}/offers{?cursor,limit,order}

**Parameters**

- **address** (*str*) – The account ID to retrieve offers from
- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.

**Returns** The list of offers for an account in a JSON response.

**Return type** dict

**account_operations**(*address*, *params=None*, *sse=None*, ***kwargs*)
This endpoint represents all operations that were included in valid transactions that affected a particular account.

GET /accounts/{account}/operations{?cursor,limit,order}

**Parameters**

- **address** (*str*) – The account ID to list operations on
- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.
- **sse** (*bool*) – Use server side events for streaming responses

**Returns** The list of operations for an account in a JSON response.

**Return type** dict

**account_payments**(*address*, *params=None*, *sse=None*, ***kwargs*)
This endpoint responds with a collection of Payment operations where the given account was either the sender or receiver.

GET /accounts/{id}/payments{?cursor,limit,order}

**Parameters**

- **address** (*str*) – The account ID to list payments to/from
- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.
- **sse** (*bool*) – Use server side events for streaming responses

**Returns** The list of payments for an account in a JSON response.

**Return type** dict

**account_transactions**(*address*, *params=None*, *sse=None*, ***kwargs*)
This endpoint represents all transactions that affected a given account.

GET /accounts/{account_id}/transactions{?cursor,limit,order}

**Parameters**

- **address** (*str*) – The account ID to list transactions from

- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.

**Returns** The list of transactions for an account in a JSON response.

**Return type** dict

**assets**(*params=None*, ***kwargs*)

This endpoint represents all assets. It will give you all the assets in the system along with various statistics about each.

See the documentation below for details on query parameters that are available.

GET /assets{?asset_code,asset_issuer,cursor,limit,order}

**Parameters** **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.

**Returns** A list of all valid payment operations

**Return type** dict

**effects**(*params=None*, *sse=None*, ***kwargs*)

This endpoint represents all effects.

GET /effects{?cursor,limit,order}

**Parameters**

- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.
- **sse** (*bool*) – Use server side events for streaming responses

**Returns** A list of all effects

**Return type** dict

**ledger**(*ledger_id*, ***kwargs*)

The ledger details endpoint provides information on a single ledger.

GET /ledgers/{sequence}

**Parameters** **ledger_id** (*int*) – The id of the ledger to look up

**Returns** The details of a single ledger

**Return type** dict

**ledger_effects**(*ledger_id*, *params=None*, ***kwargs*)

This endpoint represents all effects that occurred in the given ledger.

GET /ledgers/{id}/effects{?cursor,limit,order}

**Parameters**

- **ledger_id** (*int*) – The id of the ledger to look up
- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.

**Returns** The effects for a single ledger

**Return type** dict

**ledger_operations**(*ledger_id*, *params=None*, ***kwargs*)

This endpoint returns all operations that occurred in a given ledger.

GET /ledgers/{id}/operations{?cursor,limit,order}

**Parameters**

- **ledger_id** (*int*) – The id of the ledger to look up
- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.

**Returns** The operations contained in a single ledger

**Return type** dict

**ledger_payments**(*ledger_id*, *params=None*, ***kwargs*)
This endpoint represents all payment operations that are part of a valid transactions in a given ledger.

GET /ledgers/{id}/payments{?cursor,limit,order}

**Parameters**

- **ledger_id** (*int*) – The id of the ledger to look up
- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.

**Returns** The payments contained in a single ledger

**Return type** dict

**ledger_transactions**(*ledger_id*, *params=None*, ***kwargs*)
This endpoint represents all transactions in a given ledger.

GET /ledgers/{id}/transactions{?cursor,limit,order}

**Parameters**

- **ledger_id** (*int*) – The id of the ledger to look up.
- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.

**Returns** The transactions contained in a single ledger

**Return type** dict

**ledgers**(*params=None*, *sse=None*, ***kwargs*)
This endpoint represents all ledgers.

GET /ledgers{?cursor,limit,order}

**Parameters** **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.

**Returns** All ledgers on the network.

**Return type** dict

**operation**(*op_id*, ***kwargs*)
The operation details endpoint provides information on a single operation.

GET /operations/{id}

**Parameters** **op_id** (*id*) – The operation ID to get details on.

**Returns** Details on a single operation

**Return type** dict

**operation_effects**(*op_id*, *params=None*, ***kwargs*)
This endpoint represents all effects that occurred as a result of a given operation.

GET /operations/{id}/effects{?cursor,limit,order}

**Parameters**

- **op_id** (*int*) – The operation ID to get effects on.
- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.

**Returns** A list of effects on the given operation

**Return type** dict

**operations**(*params=None*, *sse=None*, ***kwargs*)
This endpoint represents all operations that are part of validated transactions.

GET /operations{?cursor,limit,order}

**Parameters**

- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.
- **sse** (*bool*) – Use server side events for streaming responses

**Returns** A list of all operations

**Return type** dict

**order_book**(*params=None*, ***kwargs*)
Return, for each orderbook, a summary of the orderbook and the bids and asks associated with that orderbook.

See the external docs below for information on the arguments required.

GET /order_book

**Parameters** **params** (*dict*) – The query parameters to pass to this request.

**Returns** A list of orderbook summaries as a JSON object.

**Return type** dict

**paths**(*params=None*, ***kwargs*)
Load a list of assets available to the source account id and find any payment paths from those source assets to the desired destination asset.

See the below docs for more information on required and optional parameters for further specifying your search.

GET /paths

**Parameters** **params** (*dict*) – The query parameters to pass to this request, such as source_account, destination_account, destination_asset_type, etc.

**Returns** A list of paths that can be used to complete a payment based on a given query.

**Return type** dict

**payments**(*params=None*, *sse=None*, ***kwargs*)
This endpoint represents all payment operations that are part of validated transactions.

GET /payments{?cursor,limit,order}

**Parameters**

- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.
- **sse** (*bool*) – Use server side events for streaming responses

**Returns** A list of all valid payment operations

**Return type** dict

**submit**(*te*, ***kwargs*)
Submit a transaction to Horizon.

POST /transactions

Uses form-encoded data to send over to Horizon.

**Parameters** **te** (*bytes*) – The transaction envelope to submit

**Returns** The JSON response indicating the success/failure of the submitted transaction.

**Return type** dict

**trade_aggregations**(*params=None*, ***kwargs*)
Load a list of aggregated historical trade data, optionally filtered by an orderbook.

GET /trade_aggregations

**Parameters** **params** (*dict*) – The query parameters to pass to this request, such as start_time, end_time, base_asset_type, counter_asset_type, order, limit, etc.

**Returns** A list of collected trade aggregations

**Return type** dict

**trades**(*params=None*, ***kwargs*)
Load a list of trades, optionally filtered by an orderbook.

See the below docs for more information on required and optional parameters for further specifying your search.

GET /trades

**Parameters** **params** (*dict*) – The query parameters to pass to this request, such as base_asset_type, counter_asset_type, cursor, order, limit, etc.

**Returns** A list of trades filtered by a given query

**Return type** dict

**transaction**(*tx_hash*, ***kwargs*)
The transaction details endpoint provides information on a single transaction.

GET /transactions/{hash}

**Parameters** **tx_hash** (*str*) – The hex-encoded transaction hash

**Returns** A single transaction's details

**Return type** dict

**transaction_effects**(*tx_hash*, *params=None*, ***kwargs*)
This endpoint represents all effects that occurred as a result of a given transaction.

GET /transactions/{hash}/effects{?cursor,limit,order}

**Parameters**

- **tx_hash** (*str*) – The hex-encoded transaction hash

- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.

**Returns** A single transaction's effects

**Return type** dict

**transaction_operations**(*tx_hash*, *params=None*, ***kwargs*)
This endpoint represents all operations that are part of a given transaction.

GET /transactions/{hash}/operations{?cursor,limit,order}

**Parameters**

- **tx_hash** (*str*) – The hex-encoded transaction hash
- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.

**Returns** A single transaction's operations

**Return type** dict

**transaction_payments**(*tx_hash*, *params=None*, ***kwargs*)
This endpoint represents all payment operations that are part of a given transaction.

GET /transactions/{hash}/payments{?cursor,limit,order}

**Parameters**

- **tx_hash** (*str*) – The hex-encoded transaction hash
- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.

**Returns** A single transaction's payment operations

**Return type** dict

**transactions**(*params=None*, *sse=None*, ***kwargs*)
This endpoint represents all validated transactions.

GET /transactions{?cursor,limit,order}

**Parameters**

- **params** (*dict*) – The query parameters to pass to this request, such as cursor, order, and limit.
- **sse** (*bool*) – Use server side events for streaming responses

**Returns** The list of all transactions

**Return type** dict

stellar_base.horizon.**horizon_testnet**()
Create a Horizon instance utilizing SDF's Test Network.

stellar_base.horizon.**horizon_livenet**()
Create a Horizon instance utilizing SDF's Live Network.

CHAPTER 3

# Snippets

Find examples here.

## 3.1 Snippets

### 3.1.1 Random Key Generation

```
from stellar_base.keypair import Keypair
kp = Keypair.random()
```

### 3.1.2 Multiple Key Generation

```
from stellar_base.utils import StellarMnemonic
from stellar_base.keypair import Keypair
sm = StellarMnemonic()
secret_phrase = sm.generate()
kp0 = Keypair.deterministic(secret_phrase, index=0)
kp1 = Keypair.deterministic(secret_phrase, index=1)
kp2 = Keypair.deterministic(secret_phrase, index=2)
```

### 3.1.3 Create Account

```
from stellar_base.builder import Builder
alice_seed = 'SB4675LMYLBWMKENDBAO6ZTVPLI6AISE3VZZDZASUFWW2T4MEGKX7NEI'
bob_address = 'GCRNOBFLTGLGSYOWCYINZA7JAAAZ5CXMSNM7QUYFYOHHIEZY4R6665MA'
builder = Builder(secret=alice_seed, horizon='https://horizon-testnet.stellar.org')
builder.append_create_account_op(destination=bob_address, starting_balance=2)
builder.sign()
builder.submit()
```

### 3.1.4 Payment

```
from stellar_base.builder import Builder
alice_seed = 'SB4675LMYLBWMKENDBAO6ZTVPLI6AISE3VZZDZASUFWW2T4MEGKX7NEI'
bob_address = 'GCRNOBFLTGLGSYOWCYINZA7JAAAZ5CXMSNM7QUYFYOHHIEZY4R6665MA'
builder = Builder(secret=alice_seed, horizon='https://horizon-testnet.stellar.org') \
          .append_payment_op(destination=bob_address, amount=100, asset_code='XLM',␣
↪asset_issuer=None) \
          .add_text_memo("Hey, Stellar!")
builder.sign()
builder.submit()
```

### 3.1.5 Create Offer

```
from stellar_base.builder import Builder
alice_seed = 'SB4675LMYLBWMKENDBAO6ZTVPLI6AISE3VZZDZASUFWW2T4MEGKX7NEI'
bob_address = 'GCRNOBFLTGLGSYOWCYINZA7JAAAZ5CXMSNM7QUYFYOHHIEZY4R6665MA'
selling_code = 'XLM'
selling_issuer = None
buying_code = 'XCN'
buying_issuer = 'GCNY5OXYSY4FKHOPT2SPOQZAOEIGXB5LBYW3HVU3OWSTQITS65M5RCNY'
price = 5.5
amount = 12.5
builder = Builder(secret=alice_seed, horizon='https://horizon-testnet.stellar.org') \
          .append_manage_offer_op(selling_code, selling_issuer, buying_code, \
           buying_issuer, amount, price)
builder.sign()
builder.submit()
```

CHAPTER 4

# Indices

- genindex
- modindex

# Python Module Index

## S

# Index

## F



## G



## H



## I



## K



## L



## M



## N



## O

## P



## R



## S



## T



## V

# X